KfK 3384
Juli 1982

Failure Diagnosis and Fault Tree Analysis

G. Weber
Institut für Datenverarbeitung in der Technik
Projekt Nukleare Sicherheit

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Datenverarbeitung in der Technik

Projekt Nukleare Sicherheit

KfK 3384

Failure Diagnosis and Fault Tree Analysis

G. Weber

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
ISSN 0303-4003

Abstract

With the increased complexity of many current systems, safety and reliability considerations are becoming increasingly importantant. Various methods and techniques employed for design, construction and operation of nuclear reactors, reprocessing plants, chemical plants etc. lead to more safety and reliability. This is due to a great extent to an increase of reliability and maintainability on the component level. However, this increase may be offset by a considerable complexity of the system. Here methods of reliability engineering are required. A systematic approach to the problems is needed. Thus reliability engineering uses a number of strategies, among them the techniques of reliable design (e.g. redundancy) and techniques of failure diagnosis (e.g. automatic search for failed units).

In this report a methodology of failure diagnosis for complex systems is presented. Systems which can be represented by fault trees are considered. This methodology is based on switching algebra, failure diagnosis of digital circuits and fault tree analysis. Relations between these disciplines are shown. These relations are due to Boolean algebra and Boolean functions used throughout. It will be shown on this basis that techniques of failure diagnosis and fault tree analysis are useful to solve the following problems:

- Describe an efficient search of all failed components if the system is failed.
- Describe an efficient search of all states which are close to a system failure if the system is still operating.

The first technique will improve the availability, the second the reliability and safety.

For these problems, the relation to methods of failure diagnosis for combinational circuits is required. Moreover, the techniques are demonstrated for a number of systems which can be represented by fault trees.

Fehlerdiagnose und Fehlerbaumanalyse

Zusammenfassung

Mit der steigenden Komplexität von zahlreichen Systemen sind heutzutage Sicherheits- und Zuverlässigkeitsüberlegungen von steigender Bedeutung. Verschiedene Methoden und Techniken, die bei Entwurf, Konstruktion und Betrieb von Reaktoren, Wiederaufarbeitungsanlagen, chemischen Anlagen u.s.w. eingesetzt werden, ergeben mehr Sicherheit und Zuverlässigkeit. Dies ist insbesondere auf eine Erhöhung von Zuverlässigkeit und Instandhaltbarkeit auf der Komponentenebene zurückzuführen. Jedoch können diese Verbesserungen durch eine erhebliche Komplexität des Systems zumindest abgeschwächt werden. Darum sind Methoden der Zuverlässigkeitssicherung erforderlich. Eine systematische Behandlung dieser sicherheitsrelevanten Probleme ist notwendig. So verwendet die Zuverlässigkeitssicherung eine Anzahl von Strategien. Typische Beispiele sind die Behandlung von Zuverlässigkeitsfragen beim Entwurf (z.B. Verwendung von Redundanz) und der Einsatz von Fehlerdiagnose (z.B. automatische Erkennung von ausgefallenen Einheiten). In diesem Bericht soll eine Methodologie der Fehlerdiagnose für komplexe Systeme dargestellt werden. Die Methoden sind anwendbar auf Systeme, die durch Fehlerbäume dargestellt werden können. Die Methologie beruht auf Überlegungen aus Schaltalgebra, Fehlerdiagnose von digitalen Schaltnetzen und Fehlerbaumanalyse. Die Beziehungen zwischen diesen Disziplinen werden aufgezeigt. Die Beziehungen beruhen insbesondere auf der Boole'schen Algebra und den Boole'schen Funktionen, die im ganzen Bericht verwendet werden.

Es kann auf dieser Basis gezeigt werden, daß Techniken der Fehlerdiagnose und Fehlerbaumanalyse nützlich sind, folgende Probleme zu behandeln:

- Eine effiziente Suche aller ausgefallenen Komponenten (wenn das System ausgefallen ist), soll ausgeführt werden.
- Eine effiziente Suche aller Zustände, die in der Nähe eines Systemausfalls sind (wenn das System noch intakt ist), soll ausgeführt werden.

Die erste Technik wird die Verfügbarkeit erhöhen, die zweite die Zuverlässigkeit und Sicherheit.

Für diese Probleme ist die Beziehung zu Methoden der Fehlerdiagnose kombinatorischer Schaltnetze erforderlich. Die angeführten Techniken werden für eine Anzahl von Systemen demonstriert, die durch Fehlerbäume dargestellt sind.

F i g u r e s

1. Major gate types
2. Karnaugh-map
3. Parity bit generator
4. Combinational Circuit
5. Graphic Representation
6. Redundant Circuit
7. Example (Top down)
8. Example (Bottom up)
9. Illustrative Example of Fault Tree
10. Cubical Representation
11. AND-Gate with s-a-1 fault
12. Bridge Fault
13. Combinational Circuit
14. Fault Table
15. Simplified Fault Table
16. Combinational Circuit
17. Combinational Circuit
18. Irredundant Circuit
19. Network
20. System S_1
21. System S_2
22. AND-OR-network
23. AND-OR-network
24. Karnaugh-map
25. Schematic Diagram of Device
26. Fault Tree
27. Standby System
28. Fault Tree

29. Residual Heat Removal Fault Tree

30. Block Diagram for Nitric Acid Cooler

31. Input-output Models

32. Flow Diagram for Nitric Acid Cooler Process

33. Fault Tree for Nitric Acid Cooler

34. Subtree

35. Illustrative Example of Fault Tree

CONTENTS

0.	Introduction	1
1.	Introduction to Switching Algebra	2
2.	Introduction to Failure Diagnosis	41
3.	Fault Trees	53
4.	Diagnosis Procedures	61
5.	Tests for two Types of Faults	69
6.	Examples with Various Fault Trees	83
	Reference	111

INTRODUCTION

The design, construction and operation of complex systems (nuclear reactors, reprocessing plants, chemical plants etc.) has to meet requirements regarding safety, reliability and availability. Here methods of reliabiliy engineering are required. A systematic approach to these problems is needed. Thus reliability engineering uses a number of strategies, among them the techniques of reliable design (e.g. static and dynamic redundancy) and techniques of failure diagnosis (e.g. automatic search for failed units, design for diagnosability).
In this report a methodology for complex systems is presented. Systems which can be represented by fault trees are considered. The following subjects are significant for our approach:
Concepts of switching algebra including some questions of representation. This leads to the representation by prime implicants and min cuts (sect. 1). Basic concepts of failure diagnosis are introduced (sect. 2). Concepts of fault tree analysis are introduced: coherence, min cuts (sect. 3). Diagnosis procedures are introduced which may be applied to systems represented by fault trees: A test which leads to a prompt failure diagnosis for a failed system. A test which finds all states adjacent to system failure. The first test increases availability, the second test increases safety (sect. 4).
Then a discussion of the corresponding concepts of failure diagnosis of combinational circuits is given (sect. 5). Finally, a number of examples demonstrates the use of the introduced methods for nuclear and other technologies. Results of diagnosis and conclusions on the efficiency of test methods are presented (sect. 6).

While all methods mentioned have been used extensively either for computer science or for safety questions a unified approach was not yet available.

1. Introduction to Switching Algebra

1.1 Basic Concepts

1.2 Basic Properties

1.3 Switching Functions

1.4 Representations of Boolean Expressions

1.5 Prime Implicants and Coverage

1.6 Methods to obtain Prime Implicants

1.7 Algorithms to find a simplified sum-of-products Representation

1.8 Cubical Representation of Boolean Functions

1. Introduction to Switching Algebra

We give some basic concepts for switching algebra. This technique is closely related to Boolean algebra. It is useful for

- failure diagnosis and
- fault tree analysis.

1.1 Basic concepts

We assume the existence of a two-valued switching-variable "x" which can assume the values 0 and 1. (Note, that 0, 1 are not the real numbers.) No other values are possible here.

A switching algebra is an algebraic system consisting of the set $\{0,1\}$, two binary operations called 'disjunction'(inclusive OR), 'conjunction' (AND), and one unary operation called 'negation' (NOT).

We write + ($\vee$) for OR, $\cdot$ ($\wedge$) for AND, - for NOT.

The definitions of the following relations (AND, OR, NOT, etc.) are given in Fig. 1 (see /1/).

All the gate definitions exept NOT can easily be generalized to allow any input number.

A set G of gate types is called 'complete' if any combinational function can be realized by a circuit that contains gates from G only. Examples of complete sets are {NAND} , {NOR} , {AND, NOT}, { OR, NOT } , {AND, OR, NOT} .

We use the set {AND, OR, NOT} as complete set G.

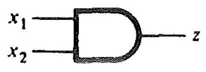
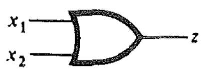
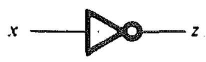
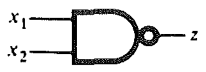
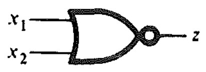
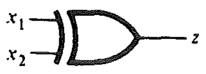
Name	Circuit symbol	Truth table	Equation
AND		x_1 x_2 | z 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	$z = x_1 x_2$ or $z = x_1 \wedge x_2$
OR		x_1 x_2 | z 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	$z = x_1 + x_2$ or $z = x_1 \vee x_2$
NOT		x | z 0 | 1 1 | 0	$z = \bar{x}$
NAND		x_1 x_2 | z 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0	$z = \overline{x_1 x_2}$
NOR		x_1 x_2 | z 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0	$z = \overline{x_1 + x_2}$
EXCLUSIVE-OR		x_1 x_2 | z 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0	$z = x_1 \oplus x_2$

Fig. 1 Major Gate Types

1.2 Basic Properties

We mention a few basic properties of switching algebra. They are also sufficient for a set of axioms. Note that there are also other sets of axioms.

Let x,y,z,... be variables. Then we use the following pairs of identities:

Idempotency

$x + x = x$

$x \cdot x = x$

Note the difference from arithmetic where no idempotency law exists.

Commutativity

$x + y = y + x$

$x \cdot y = y \cdot x$

Associativity

$(x + y) + z = x + (y + z)$

$(x \cdot y) \cdot z = x \cdot (y \cdot z)$

Distributivity

$x \cdot (y + z) = x \cdot y + x \cdot z$

$x + y \cdot z = (x + y)(x + z)$

Note the difference from arithmetic.

Complementation

$x + \overline{x} = 1$

$x \cdot \overline{x} = 0$

Note the difference from arithmetic.

From the Basic Concepts (1.1) and Basic Properties we can deduce many theorems. Two important theorems are the theorems of De Morgan:

$\overline{x + y} = \overline{x} \cdot \overline{y}$

$\overline{x \cdot y} = \overline{x} + \overline{y}$

We can use truth-tables to prove De Morgan's theorems:

x	y	$\overline{x}$	$\overline{y}$	$x + y$	$\overline{x + y}$	$\overline{x}\,\overline{y}$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

Note

For n variable we can write

$$\left.\begin{aligned}&(a)\quad \sum_{i=1}^{n} x_i = x_1 + x_2 + \dots + x_n\\ &\text{or}\quad \bigvee_{i=1}^{n} x_i = x_1 \vee x_2 \vee \dots \vee x_n\end{aligned}\right\}\quad \text{Disjunction, sum-term}$$

$$\left.\begin{aligned}&(b)\quad \prod_{i=1}^{n} x_i = x_1\, x_2 \cdots x_n\\ &\text{or}\quad \bigwedge_{i=1}^{n} x_i = x_1 \wedge x_2 \wedge \cdots \wedge x_n\end{aligned}\right\}\quad \text{Conjunction, Boolean monomial, product-term}$$

This can be represented by an AND-gate or OR-gate with n inputs.

1.3 Switching Functions

We introduce the concept of switching function, extending the switching algebra to functions of binary variables. The switching function is a Boolean function. Thus it is clearly related to the structure function.

Def.: A 'switching function'

$$f\ (x_1, x_2, \dots, x_n)$$

of n two-valued variables $x_1, x_2, \dots, x_n$ ($x_1 = 0,1$) is a correspondence which assigns for each of the 2^n combinations one value of $\{0,1\}$.

The switching function can be represented using
(a) a truth table
(b) maps
(c) graphic representations, diagrams, fault trees
(d) Boolean expressions, structure functions.
Clearly, for a high number of variables, (a) and (b) become extremely large. E.g. we will habe for n variables 2^n rows in the truth table.

Example:

We will introduce all representations (a) - (d) for an example:

(a) Truth table
A parallel parity-bit generator /2/: This unit must produce an output 1 if and only if an odd number of its inputs have value 1. Take the example of three-bit code words, i.e. the circuit has three inputs x_1, x_2, x_3 and its output f must be equal to 1 if 1 or 3 of the inputs are 1. We can immediately construct the truth table:

row	x_1	x_2	x_3	f	Number of inputs = 1
0	0	0	0	0	0, even
1	0	0	1	1	1, odd
2	0	1	0	1	1, odd
3	0	1	1	0	2, even
4	1	0	0	1	1, odd
5	1	0	1	0	2, even
6	1	1	0	0	2, even
7	1	1	1	1	3, odd

Table I, truth table

(b) Map

We give a map-representation, based on the truth table.

x_1 x_2	0 0	0 1	1 1	1 0
x_3 0	0	1	0	1
1	1	0	1	0

Fig. 2: Map

(c) Graphic Representation

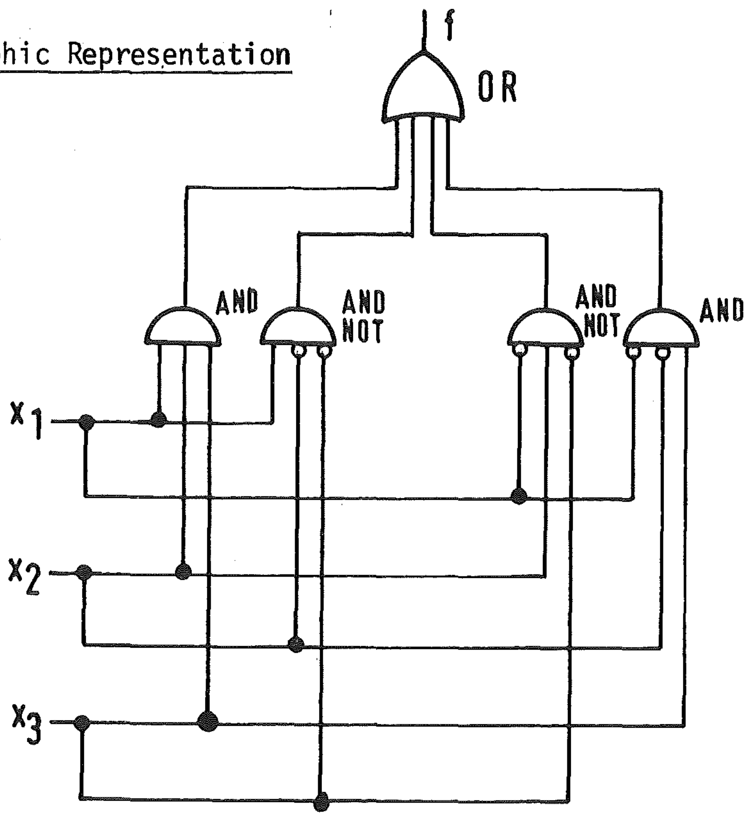


Fig. 3: parallel parity-bit generator

(d) Boolean Representation (Boolean polynomial).

For this circuit we get as Boolean representation:

$$f\ (x_1, x_2, x_3) \quad = \quad x_1\ x_2\ x_3 \vee x_1\ \overline{x}_2\ \overline{x}_3 \vee \overline{x}_1\ x_2\ \overline{x}_3 \vee \overline{x}_1\ \overline{x}_2\ x_3$$

i.e. $f = 1$ if either x_1 and x_2 and x_3 are $= 1$ or exactly one input is $= 1$.

It is possible to represent switching functions using different techniques. Each will lead to the same truth table.

Canonical forms

We recall that truth-tables are a means for representing switching functions (Boolean functions). We also mentioned that Boolean expressions may be written as Boolean polynomials. Now we give some considerations which are

- closely related to truth tables and are
- easily generalized for switching algebra (Boolean algebra).

Assume, we have a function $f\ (x_1, x_2, \ldots, x_n)$ represented in a truth table. Then we get two representations which are called 'canonical forms' which will be discussed next:

- the disjunctive normal form (dnf)
- the conjunctive normal form (cnf).

Disjunctive normal form

We introduce the concept of 'minterm'. A minterm is a conjunction (product) of n variables:

$$p\ (x_1, x_2, \cdot\cdot\cdot, x_n)$$

Each variable may be either complemented or uncomplemented. The characteristic property of a minterm is that it assumes the value 1 for exactly one combination of the variables.

Then we can write any Boolean function as a disjunction of minterms, called disjunctive normal form (dnf):

$$f\ (x_1, x_2, \cdots, x_n) = \bigvee_{i=o}^{2^n-1} c_i\ p_i\ (x_1, x_2, \cdots, x_n)$$

where the constant c_i is defined as follows:

$c_i = 1$ denotes the minterms which in a disjunction generate the function $f\ (x_1, x_2, \cdots, x_n)$, $c_i = 0$ is related to all other minterms.

Relation to truth-table

For each row j where $f\ (x_1, x_2, \cdots, x_n) = 1$, we get a minterm

$$p_j\ (x_1, x_2, \cdots, x_n)$$

If in this row we have

$$\begin{array}{llll} x_1 = 0 & & \overline{x_i} & \\ & \text{we write} & & \text{in } p_j\ (x_1, x_2, \cdots, x_n). \\ x_2 = 1 & & x_i & \end{array}$$

Now we get a disjunction of minterms $p_j\ (x_1, x_2, \cdots, x_n)$

which is equal to the given function $f\ (x_1, x_2, \cdots, x_n)$:

$$f\ (x_1, x_2, \cdots, x_n) = \bigvee_{j\epsilon r_1} p_j\ (x_1, x_2, \cdots, x_n)$$

where j goes over all rows where f = 1, i.e. r_1 is the set of all row-numbers where f = 1.

Note:

$$f\ (x_1, x_2, \cdots, x_n) = \bigvee_{i=o}^{2^n-1} c_i\ p_i\ (x_1, x_2, \cdots, x_n) = \bigvee_{j\epsilon r_1} p_j(x_1, x_2, \cdots, x_n)$$

<u>Example</u>

We refer again to Table I (parity-bit generator).
It can be seen that $f\ (x_1, x_2, x_3) = 1$ for the set r_1, $r_1 = \{1,2,4,7\}$.

We get the minterms p_j:

				Decimal notation
row 1:	0 0 1	$\longleftrightarrow$	$p_1 = \bar{x}_1\ \bar{x}_2\ x_3$	1
row 2:	0 1 0	$\longleftrightarrow$	$p_2 = \bar{x}_1\ x_2\ \bar{x}_3$	2
row 4	1 0 0	$\longleftrightarrow$	$p_4 = x_1\ \bar{x}_2\ \bar{x}_3$	4
row 7	1 1 1	$\longleftrightarrow$	$p_7 = x_1\ x_2\ x_3$	7

The disjunctive normal form is

$$\bigvee_{j\in r_1} p_j = \bar{x}_1\ \bar{x}_2\ x_3 \vee \bar{x}_1\ x_2\ \bar{x}_3 \vee x_1\ \bar{x}_2\ \bar{x}_3 \vee x_1\ x_2\ x_3$$

or, in decimal notation $f\ (x_1, x_2, x_3) = \sum(1, 2, 4, 7)$

<u>Note:</u>

1. This form is a sum-of-products-form (sop), if we consider the disjunction as sum, the conjunction as product, a special form of a Boolean polynomial.

2. There are some noteworthy properties of the dnf: There is only <u>one</u> dnf for a given Boolean function $f\ (x_1, x_2, \ldots, x_n)$, equivalent to the unique truth table.

3. All terms are disjoint, $p_j \cdot p_k = 0$ for $j \neq k$.

   Assume the contrary, i.e. each variable of p_j which is uncomplemented (complemented) in p_j must also be uncomplemented (complemented) in p_j. Thus p_j and p_k cannot be different, i.e. $j = k$.

Relation to Boolean Expressions

To obtain the disjunctive normal form for any given Boolean function a simple procedure can be used. This procedure will also be useful for further considerations. It can be shown that this procedure always leads to a result /1/.

Step 1: Expand the given function to a sum of products form which needs no brackets.

Step 2: Examine each product term. If it is a minterm, retain it, and continue to the next term.

Step 3: In each product which is not a minterm check the variables that do not occur. For each x_i that does not occur multiply the product by $(x_i + \overline{x}_i)$.

Step 4: Multiply out all products and eliminate redundant terms.

Example

Determine the dnf for the following function

$$f\ (x,\ y,\ z) = \overline{x}_3 + x_2\ (\overline{x}_1 + x_1\ x_3)$$

This function could be represented graphically as follows:

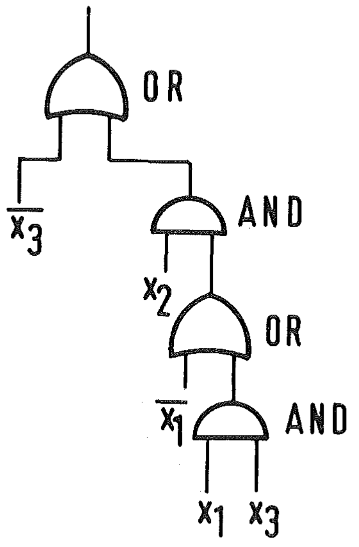


Fig. 4 Combinational Circuit

Step 1:

$$f(x_1, x_2, x_3) = \overline{x}_3 + \overline{x}_1 x_2 + x_1 x_2 x_3$$

Step 2:

$x_1 x_2 x_3$ is a minterm

Step 3:

$$f(x_1, x_2, x_3) = \overline{x}_3 (x_2 + \overline{x}_2)(x_1 + \overline{x}_1) + \overline{x}_1 x_2 (x_3 + \overline{x}_3) + x_1 x_2 x_3$$

Step 4:

$$f(x_1, x_2, x_3) = x_1 x_2 \overline{x}_3 + x_1 \overline{x}_2 \overline{x}_3 + \overline{x}_1 x_2 \overline{x}_3 + \overline{x}_1 \overline{x}_2 \overline{x}_3 + \overline{x}_1 x_2 x_3 + x_1 x_2 x_3$$

Note

A similar discussion is possible for 'maxterms'. A maxterm is a disjunction (sum) of n variables. The conjunction of maxterms is called a conjunctive normal form. This will not be of much use for problems discussed here.

1.4 Representations of Boolean Expressions

It is useful to have several alternative representations for Boolean expressions /3/. Assume again we have a switching function given as follows:

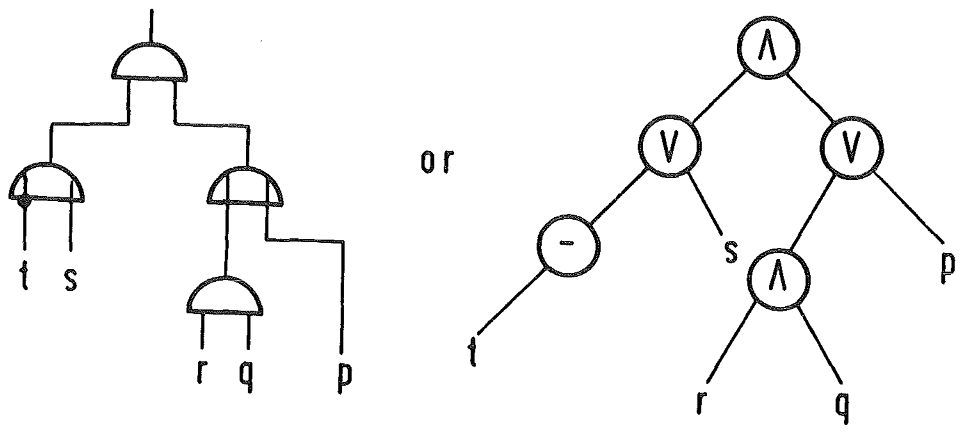


Fig. 5 Graphic Representation

The tree representation (Fig. 5b) is probably most graphic, we can easily see the predecessors, sucessors etc. We can write the expression also using the usual Boolean operations,

$$f = ((q \wedge r) \vee p) \wedge (s \vee \overline{t})$$

We also want to introduce a representation which will be needed for some methods (as discussed in 1.7.). We introduce the following notation for Fig. 5b:

For branches between vertices we give

1 if the branch goes to the right
2 if the branch goes to the left.

Thus we get (Fig. 5c):

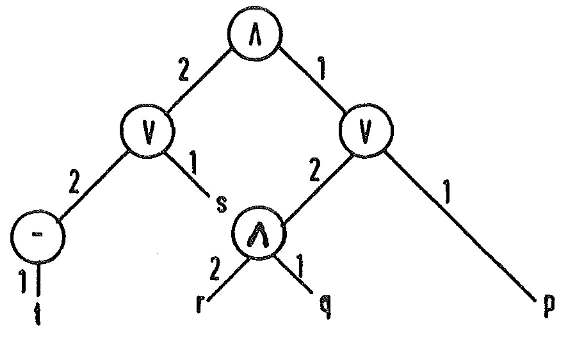


Fig. 5c

E.g. for r we can write 122, for the gate $r \wedge p$ we can write 12 (as 'coordinates'). We can represent the tree as a data structure, called the 'full left list matrix' /3/:

Here in collumn 1 is the number of predecessors,
in collumn 2 the type of operator or operand,
in collumn 3,4,··· the numbers giving 'coordinates'.

Collumn	1	2	3	4	5
	2	∧			
	2	v	1		
	0	p	1	1	
	2	∧	1	2	
	0	q	1	2	1
	0	r	1	2	2
	2	v	2		
	0	s	2	1	
	1	-	2	2	
	0	t	2	2	1

It is sometimes convenient, to simplify the full left list matrix to a left list matrix, dropping the coordinates:

collumn	1	2
	2	∧
	2	v
	0	p
	2	∧
	0	q
	0	r
	2	v
	0	s
	1	-
	0	t

It can be shown that, if a binary relation such as $\wedge$, is written in front of its two operands in the form $\wedge$ x y (instead of $x \wedge y$), then by sonsistent use of such a notation ('prefix notation') no parentheses are necessary. As polish equivalents of Boolean connectives, we get (/3/, /4/):

Boolean	Polish	Reverse Polish
$\overline{x}$	-,x	x,-
$x \wedge y$	$\wedge$, x, y	x, y, $\wedge$
x v y	v, x, y	x, y, v
$x \oplus y$	$\oplus$, x, y	x, y, $\oplus$

Thus our tree may be written in a Lucasiewicz- or parenthesis-free-notation (also called Polish notation):

(a) Polish Notation, prefix notation

(∧, v, p, ∧, q, r, v, s, -, t)

(b) Reverse Polish Notation, postfix notation

(t, -, s, v, q, r, ∧, p, v, ∧)

Note:

- The reverse Polish notation requires that the operators are written in reverse order. Since all operators needed here are related to commutative operations, the order of the variables is not affected.
- If the operators cover more than two variables this should be indicated, e.g. $x \wedge y \wedge z$ can be written $\wedge$(3), x, y, z.

It will be seen in sect. 1.7 how the left list matrix and the reverse polish notation is of direct relevance to problems of switching theory and fault tree analysis.

1.5 Prime Implicants and Coverage

A switching function $f(x_1, x_2, \ldots, x_n)$ is said to cover another function $g(x_1, x_2, \cdots, x_n)$, denoted

$$f \supseteq g$$

if f assumes the value 1 whenever g does. Thus, if f covers, then it has a 1 in every row in the truth table in which g has a 1.

Example:

Let $f = x_1 \oplus x_2$ (Exclusive OR)

$g_1 = x_1 \bar{x}_2$, $g_2 = \bar{x}_1 x_2$

x_1	x_2	f
0	0	0
1	0	1
0	1	1
1	1	0

x_1	x_2	g_1	g_2
0	0	0	0
1	0	1	0
0	1	0	1
1	1	0	0

Thus: $f \supseteq g_1$ and $f \supseteq g_2$

If f covers g and g covers f, then f and g are equivalent.

Example:

Let $f = x_1 \oplus x_2$ and

$g = g_1 \vee g_2$

Then f and g are equivalent.

Let $f(x_1, x_2, \ldots, x_n)$ be a switching function and $h(x_1, x_2, \cdots, x_n)$ be a product of literals (conjunction). If f covers h, then h is said to imply f, or h is said to be an implicant of f. The implicant is denoted $h \rightarrow f$.

Example: g_1 and g_2 are implicants of f.

Definition: A prime implicant p of a function f is a product term which is covered by f such that the deletion of any literal from p results in a new product which is not covered by f. In other words: p is a prime implicant if and only if p implies f but does not imply any product with fewer literals which also implies f. The set of all prime implicants will be denoted $\{p_i\}$.

Example:
$\bar{x}\,y$ is a prime implicant of

$$f = \bar{x}\,y + x\,z + \bar{y}\,\bar{z}$$

since it is covered by f but neither $\bar{x}$ nor y alone implies f.

A combinational circuit is 'redundant' if it is possible to remove lines and/or gates in such a way that the resulting circuit is equivalent. A combinational circuit which is not redundant, will be called irredundant.

Example:

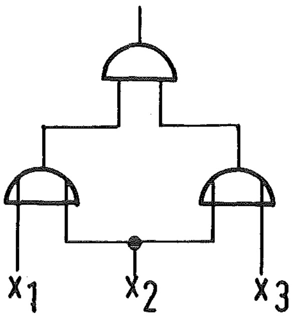


Fig. 6 Redundant Circuit

This circuit is redundant.

$$f = (x_1 + x_2)\,(x_2 + x_3) = x_1\,x_3 + x_2$$

Every circuit may be represented as a sum-of-products form /1/.

Theorem: Every irredundant sum-of-products (sop) equivalent to f is a union of prime implicants of f:

$$f = \bigvee_{i=1}^{\ell} p_i$$

Proof: Let f^* be an irredundant sop-expression equivalent to f, and suppose that f contains a product term p which is not a prime implicant. Since p is not a prime implicant, it is possible to replace it with another product term which consists of fewer literals. Hence f contains redundant literals, which contradicts our initial assumption. □

1.6 Methods to obtain Prime Implicants

We discuss some methods to obtain prime implicants. Many methods use explicitly the representation of Boolean functions by min-terms. This is true e.g. for the Quine-Mc Cluskey method and others, given in the literature e.g. /5/, /6/.

It seems to be more important, to have a method which may be used for a Boolean function represented without using min-terms. This will also prove useful for fault trees /7/.

Nelson's Algorithm

The following remarks are in order:

- F is a Boolean function which already has been transformed into a sum-of-products form.
- If in this algorithm a Boolean expression E is 'complemented', this means not only applying the complement to the expression, but also repeatedly using De Morgan's rules, i.e.

  $E = x\,y + \bar{y}\,z$ leads to

  $\bar{E} = \overline{x\,y + \bar{y}\,z} = \overline{x\,y} \cdot \overline{\bar{y}\,z} = (\bar{x} + \bar{y})\,(y + \bar{z})$

Algorithm 1

Step 1 Complement F.
Obtain $\bar{F}$ applying De Morgan's rules.
Expand $\bar{F}$ into a disjunctive normal form.
Drop zero products $(x\,\bar{x} = 0)$,
repeated literals $(x\,x = x)$,
make absorptions $(x + xy = x)$.
This result is $\bar{\Phi}$.

Step 2 Complement $\bar{\Phi}$
Obtain Φ applying De Morgan's rules.
Expand Φ into a disjunctive normal form.
Drop zero products,
repeated literals,
make absorptions.
The result is $\sum_i p_i$, the sum of all prime implicants, and only of prime implicants. □

Example:

Step 1 $F = x_1 x_2 + \bar{x}_2 x_3 x_4 + x_3 \bar{x}_4$

Complement:

$\bar{F} = (\bar{x}_1 + x_2)(x_2 + \bar{x}_3 + \bar{x}_4)(\bar{x}_3 + x_4)$

Expand and simplify:

$\bar{\Phi} = \bar{x}_1 x_2 x_4 + \bar{x}_1 \bar{x}_3 + \bar{x}_2 \bar{x}_3$

Step 2 Complement

$\Phi = (x_1 + \bar{x}_2 + \bar{x}_4)(x_1 + x_3)(x_2 + x_3)$

Expand and simplify:

$\sum p_i = x_1 x_2 + x_1 x_3 + \bar{x}_2 x_3 + x_3 \bar{x}_4$

It is often useful to simplify the Boolean functions needed in Algorithm 1 by factoring.

Example:

$F = x_1 x_2 + \bar{x}_2 x_3 x_4 + x_3 \bar{x}_4$

may be rewritten (factored) as

$F = x_1 x_2 + x_3 (\bar{x}_2 x_4 + \bar{x}_4)$

Then the algorithm may be done with a considerable amount of saving operations /8/.

Algorithm 2 (with factoring)

Step 1 Factor anywhere possible in F.
Complement F.
Obtain $\bar{F}$ applying De Morgan's rules.
Expand $\bar{F}$ into a disjunctive normal form.
Drop zero products (xx = 0),
repeated literals (xx = x),
make absorptions (x + x y = x).

The result is $\bar{\Phi}$.

Step 2 Factor anywhere possible in $\overline{\Phi}$.
Complement $\overline{\Phi}$.
Obtain Φ applying De Morgan's rules.
Expand Φ into a disjunctive normal form.
Drop zero products,
repeated literals,
make absorptions.
The results is $\sum_i p_i$, the sum of all prime implicants, and only of prime implicants.□

Example:

Step 1 $F = x_1 x_2 + \bar{x}_2 x_3 x_4 + x_3 \bar{x}_4$

Factor: $F = x_1 x_2 + (\bar{x}_2 x_4 + \bar{x}_4) x_3$

Complement:

$$\bar{F} = (\bar{x}_1 + \bar{x}_2)(x_2 + \bar{x}_4) x_4 + \bar{x}_3)$$

Expand and simlify:

$$\overline{\Phi} = \bar{x}_1 x_2 x_4 + \bar{x}_1 \bar{x}_3 + \bar{x}_2 \bar{x}_3$$

Step 2

Factor: $\overline{\Phi} = \bar{x}_1 (x_2 x_4 + \bar{x}_3) + \bar{x}_2 \bar{x}_3$

Complement:

$$\Phi = \overline{\bar{x}_1 (x_2 x_4 + \bar{x}_3)} \,.\, \overline{\bar{x}_2 \bar{x}_3}$$

$$= (x_1 + (\bar{x}_2 + \bar{x}_4) x_3)(x_2 + x_3)$$

Expand and simplify:

$$\sum_i p_i = x_1 x_2 x_1 x_3 + \bar{x}_2 x_3 + x_3 \bar{x}_4$$

Notice the savings in the number of terms if F and $\overline{\Phi}$ has been factored.

1.7 Algorithms to find a simplified sum-of-products representation (sop)

The algorithms to find a simplified s-o-p-representation can be used for the Nelson-Algorithm, Algorithm 1. For some special cases, i.e. Boolean functions which can be represented using AND and OR alone (but without complements; see sect. 3.3) these algorithms even give all prime implicants /9/.

Top-Down-Algorithm (Fussell's Algorithm)

We assume a switching network represented by a logical diagram.

Algorithm 3

Step 0 Start at top A_o.

Step 1 Search for predecessors of A_i $(i = 1,2,...)$
Define predecessors of A_i.
$(A_i^1, A_i^2) = \text{pred}\ (A_i)$.

Step 2 If A_i is an OR-gate, we get

$A_i^1 + A_i^2 = A_i$, rename A_i^1, A_i^2

If A_i is an AND-gate, we get

$A_i^1 \cdot A_i^2 = A_i$, rename A_i^1, A_i^2

Step 3 Multiply out all identified terms to obtain a sum of products. If the sum-of-products contains still gates (A_i) go to step 1, else go to step 4.

Step 4 The sum-of-product expression (consisting of components) can be simplified:
Drop repeated literals,
make absorbtions. □

Example:

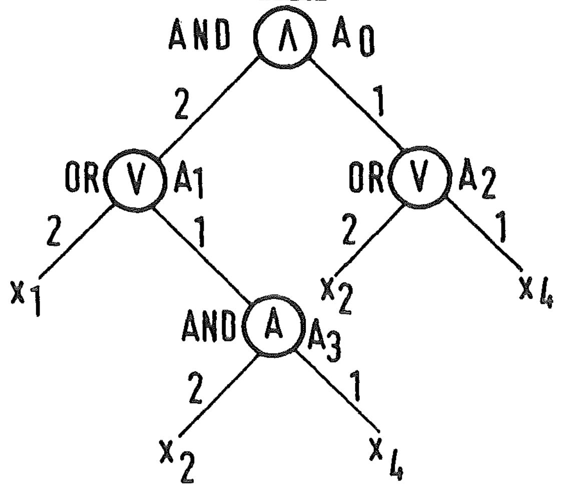


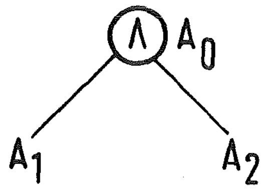
Fig. 7

This switching network can be represented in a form which contains all gates and inputs but is closer to graph theory.

Start at A_o

A_o AND-gate

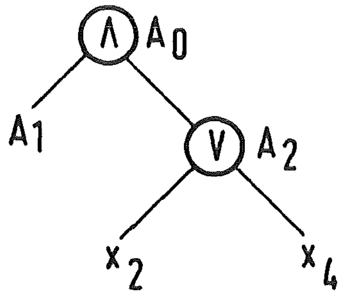
$A_1 \cdot A_2 = A_o$



A_2 OR-gate

$x_2 + x_4 = A_2$

$A_1\ x_2 + A_1\ x_4$



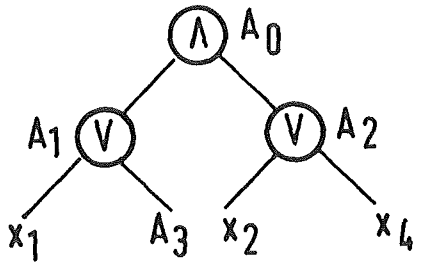
A_1 OR-gate

$x_1 + A_3 = A_1$

$(x_1 + A_3)\ x_2 + (x_1 + A_3)\ x_4$

$= \ x_1\ x_2 + A_3\ x_2$

$+ \ x_1\ x_4 + A_3\ x_4$

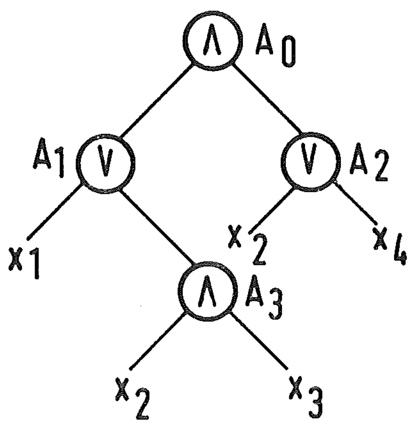


A_3 AND-gate

$x_2 \cdot x_3 = A_3$

$x_1\ x_2 + x_2\ x_3\ x_2$

$+\ x_1\ x_4 + x_2\ x_3\ x_4$



If repeated literals are dropped and if absorptions are made, we get

$\Phi = x_1\ x_2 + x_1\ x_4 + x_2\ x_3$

Bottom-Up-Algorithm (Bennett's Algorithm)

The Bottom-up-algorithm is a development of Bennett's algorithm which leads to a sum of products representation.
We recall that the reverse polish notation (left list matrix) introduced in sect. 1.4 is used /10, 11/.
We have again the tree which was also used for our top-down-algorithm.

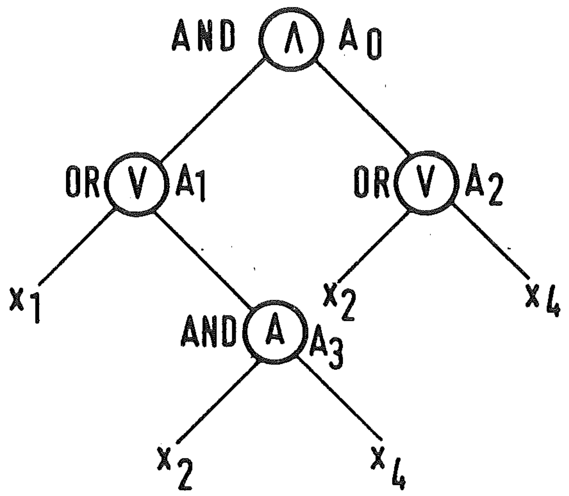


Fig. 8 Example

We can characterize all branches and thus get a full left list matrix:

A_o	2	$\wedge$	
A_2	2	$\wedge$	1
	0	x_4	1 1
	0	x_2	1 2
A_1	2	v	2
A_3	2	$\wedge$	2 1
	0	x_3	2 1 1
	0	x_2	2 1 2
	0	x_1	2 2

Full left list matrix

$(x_1, x_2, x_3, \wedge , v, x_2, x_4, v, \wedge)$

reverse polish notation

Now we describe the bottom-up-algorithm. Note, that here only AND and OR-operators are assumed. Complements are assumed to be with the variables only.

A general form of this algorithm which will be useful for large and complex trees will be discussed later /10,11/.

<u>Bottom-up-algorithm</u>

<u>Algorithm 4</u>

<u>Step 1</u> Left list matrix L given

<u>Step 2</u> Take next item from L

<u>Step 3</u> If item Operator, go to 4, else if item Operand, go to 5.

Step 4 If the operator is AND (1), withdraw the last 1 items in the list (stack) and make a conjunction, else if the operator is OR (1), withdraw the last 1 items in the list (stack) and make a disjunction.

Step 5 Push operand down into list (stack).

Step 6 Check if terms like

$$x\,\overline{x},\quad x\,x,\quad x + x\,y$$

are in the result and drop/simplify.

Step 7 Evaluate the already withdrawn terms to obtain s-o-p-expressions. Go to 2.

Step 8 If L is empty, a s-o-p-expression for the whole Boolean function is obtained. □

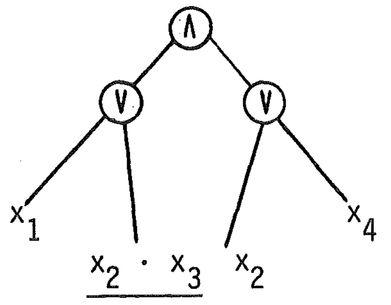
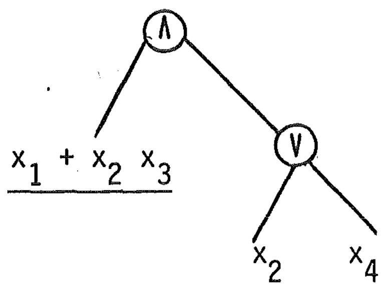
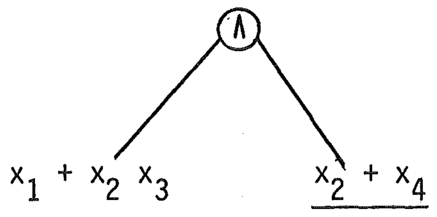
Example 1

2	$\wedge$
2	$\vee$
0	x_4
0	x_2
2	$\vee$
2	$\wedge$
0	x_3
0	x_2
0	x_1

Left list matrix L

Note: We present a number of lists (stacks) showing the mechanism of Algorithm 4, and a number of reduced trees, illustrating the bottom-up-method.

Steps 2-5

List	Operand	Reduced tree
x_3 x_2 x_1	$x_2 \cdot x_3$ x_1	
x_1	$x_1 + x_2\, x_3$	
x_4 x_2	$x_4 + x_2$ $x_1 + x_2\, x_3$	
	$x_1\, x_2 + x_1\, x_4$ $+\ x_2\, x_3\, x_2 + x_2\, x_3\, x_4$	

Steps 6-8 give

$f = x_1\, x_2 + x_1\, x_4 + x_2\, x_3$, the s-o-p-expression.

Example 2

We discuss a further example which leads to a generalization of the bottom-up algorithm. We have the following fault tree from the published literature /11,24/.

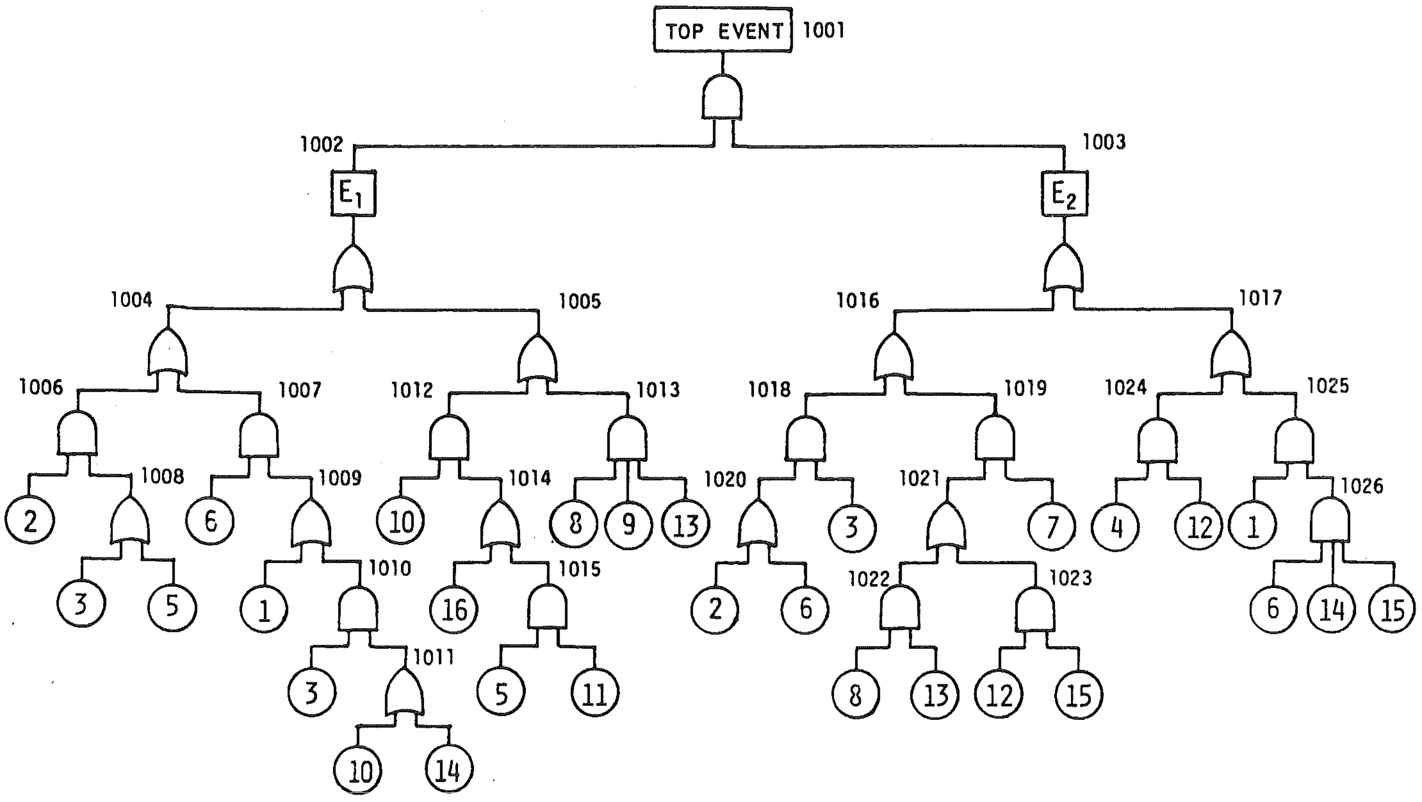


Fig. 9 Illustrative Example of Fault Tree

This fault tree is also used as an example for our section 6 (Applications of Failure Diagnosis). This fault tree is also part of the studies on hardware simulation /23/.

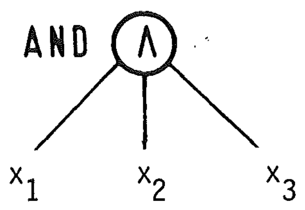
Note:

To simplify the representation of our left-list matrix, we make the following convention:

(a) Operands may be written in the same line as the operators if no ambiguity arises.

(b) If not otherwise indicated the number 1 (in $\wedge(1)$, $\vee(1)$) is equal to 2.

E. g.



is written:

3	$\wedge$
0	x_1
0	x_2
0	x_3

or more concisely $\wedge$ (3) x_1, x_2, x_3

We divide the tree into two left-lists (subtrees).

Note: For all primary events we write numbers (1,2,..,15) only.

Left-list E_1

$\vee$		
$\vee$	1	
$\wedge$ (3)	1 1	8,9,13
$\wedge$	1 2	
$\vee$	1 2 1	
$\wedge$	1 2 1 1	5,11
1 6	1 2 1 2	(Here the simplified listing does not apply)
1 0	1 2 2	
$\vee$	2	
$\wedge$	2 1	
$\vee$	2 1 1	
$\wedge$	2 1 1 1	
$\vee$	2 1 1 1 1	10,14
3	2 1 1 1 2	
1	2 1 1 2	
6	2 1 2	
$\wedge$	2 2	
$\vee$	2 2 1	3,5
2	2 2 2	

Left-list E_2

$\vee$		
$\vee$	1	
$\wedge$	1 1	
$\wedge$ (3)	1 1 1	6,14,15
1	1 1 2	
$\wedge$	1 2	4,12
$\vee$	2	
$\wedge$	2 1	
7	2 1 1	
$\vee$	2 1 2	
$\wedge$	2 1 2 1	12,15
$\wedge$	2 1 2 2	8,13
$\wedge$	2 2	
3	2 2 1	
$\vee$	2 2 2	2,6

Left-list E_1

3,5 2	$3 + 5$ 2	$2\cdot3 + 2\cdot5$ 10,14 3 1 6
(1)	(2)	(3)
$2\cdot3 + 2\cdot5$ $10 + 14$ 3 1 6	$2\cdot3 + 2\cdot5$ $3\cdot10 + 3\cdot14$ 1 6	$2\cdot3 + 2\cdot5$ $1 + 3\cdot10 + 3\cdot14$ 6
(4)	(5)	(6)
$2\cdot3 + 2\cdot5$ $1\cdot6 + 3\cdot6\cdot10+3\cdot6\cdot14$	$2\cdot3 + 2\cdot5 + 1\cdot6$ $+ 3\cdot6\cdot10 + 3\cdot6\cdot14$	5,11 16 10
(7)	(8)	(9)
$5\cdot11$ 16 10	$16 + 5\cdot11$ 10	$10\cdot16 + 5\cdot10\cdot11$
(10)	(11)	(12)
$10\cdot16 + 5\cdot10\cdot11$ 8, 9, 13	$10\cdot16 + 5\cdot10\cdot11$ $8\cdot9\cdot13$	$10\cdot16 + 5\cdot10\cdot11$ $+ 8\cdot9\cdot13$
(13)	(14)	(15)

From (8) and (15) we get

$$\Phi E_1 = 2\cdot3 + 2\cdot5 + 1\cdot6 + 3\cdot6\cdot10 + 3\cdot6\cdot14 + 8\cdot9\cdot13 + 10\cdot16 + 5\cdot10\cdot11 \qquad \text{(s-o-p-expression)}$$

Left List E_2

3 2,6	3 2 + 6	$3 \cdot 2 + 3 \cdot 6$ 7 12,15 8,13
(1)	(2)	(3)
$3 \cdot 2 + 3 \cdot 6$ 7 $12 \cdot 15$ $8 \cdot 13$	$3 \cdot 2 + 3 \cdot 6$ $7 \cdot 12 \cdot 15 + 7 \cdot 8 \cdot 13$	$3 \cdot 2 + 3 \cdot 6$ $+ 7 \cdot 12 \cdot 15 + 7 \cdot 8 \cdot 13$
(4)	(5)	(6)
6,14,15 1 4,12	6,14,15 1 $4 \cdot 12$	$6 \cdot 14 \cdot 15$ 1 $4 \cdot 12$
(7)	(8)	(9)
$1 \cdot 6 \cdot 14 \cdot 15$ $4 \cdot 12$	$1 \cdot 6 \cdot 14 \cdot 15$ $+ 4 \cdot 12$	
(10)	(11)	

From (6) and (11) we get

$$\Phi E_2 = 2 \cdot 3 + 3 \cdot 6 + 7 \cdot 8 \cdot 13 + 7 \cdot 12 \cdot 15 + 1 \cdot 6 \cdot 14 \cdot 15 + 4 \cdot 12$$

(s-o-p-expression)

Now we obtain the Boolean function for the whole tree (Fig. 9) in a few steps:

1. Allocate primary events to the set of common/non-common events;
2. Multiply Φ_{E_1} and Φ_{E_2} ;
3. Drop/simplify all terms of type $x\,\bar{x}$, $x\,x$, $x + xy$.

We introduce a technique which makes this step with a reasonable amount of calculation /11/.

1. Search for primary events which are common to E_1 <u>and</u> E_2 (c) and not common to E_1 and E_2 (non-C).

C	non-C
1	
2	
3	
	4
	5
6	
	7
8	
	9
	10
	11
	12
13	
14	
	15
	16

2. Divide primary events into the following subsets:

	Events from E_1	Events from E_2
Sets which contain only C-events	C_{1a}	C_{2a}
Sets which contain C and non-C events	C_{1b}	C_{2b}
Sets which contain only non-C events	C_{1c}	C_{2c}

3. We get for E_1 the following sets (corresponding to product terms):

$C_{11} = \{2,5\}$, $C_{12} = \{3,6,10\}$, $C_{13} = \{10,16\}$, $C_{14} = \{2,3\}$

$C_{15} = \{3,6,14\}$, $C_{16} = \{5,10,11\}$, $C_{17} = \{8,9,13\}$, $C_{18} = \{1,6\}$

Similarly, for E_2:

$C_{21} = \{3,6\}$, $C_{22} = \{4,12\}$, $C_{23} = \{1,6,14,15\}$

$C_{24} = \{7,8,13\}$, $C_{25} = \{7,12,15\}$, $C_{26} = \{2,3\}$.

4. Allocation of c_{ik} to subsets C_{1a}, C_{1b}, C_{1c} etc.

$$\{c_{14}, c_{15}, c_{18}\} \subset C_{1a}$$

$$\{c_{11}, c_{12}, c_{17}\} \subset C_{1b}$$

$$\{c_{13}, c_{16}\} \subset C_{1c}$$

a n d

$$\{c_{21}, c_{26}\} \subset C_{2a}$$

$$\{c_{23}, c_{24}\} \subset C_{2b}$$

$$\{c_{22}, c_{25}\} \subset C_{2c}$$

5. Now, each subset of E1 is related to each subset of E2.
We write for this Cartesian product:

$$C_{1a} \times C_{2a} = c_{14}\,(c_{21} \cup c_{26}) \cup c_{15}\,(c_{21} \cup c_{26}) \cup c_{18}\,(c_{21} \cup c_{26})$$

$$C_{1a} \times C_{2b} = c_{14}\,(c_{23} \cup c_{24}) \cup c_{15}\,(c_{23} \cup c_{24}) \cup c_{18}\,(c_{23} \cup c_{24})$$

$$\underline{C_{1a} \times C_{2c}} = c_{14}\,(c_{22} \cup c_{25}) \cup c_{15}\,(c_{22} \cup c_{25}) \cup c_{18}\,(c_{22} \cup c_{25})$$

$$C_{1b} \times C_{2a} = c_{11}\,(c_{21} \cup c_{26}) \cup c_{12}\,(c_{21} \cup c_{26}) \cup c_{17}\,(c_{21} \cup c_{26})$$

$$C_{1b} \times C_{2b} = c_{11}\,(c_{23} \cup c_{24}) \cup c_{12}\,(c_{23} \cup c_{24}) \cup c_{17}\,(c_{23} \cup c_{24})$$

$$\underline{C_{1b} \times C_{2c}} = c_{11}\,(c_{22} \cup c_{25}) \cup c_{12}\,(c_{22} \cup c_{25}) \cup c_{17}\,(c_{22} \cup c_{25})$$

$$C_{1c} \times C_{2a} = c_{13}\,(c_{21} \cup c_{26}) \cup c_{16}\,(c_{21} \cup c_{26})$$

$$C_{1c} \times C_{2b} = c_{13}\,(c_{23} \cup c_{24}) \cup c_{16}\,(c_{23} \cup c_{24})$$

$$C_{1c} \times C_{2c} = c_{13}\,(c_{22} \cup c_{25}) \cup c_{16}\,(c_{22} \cup c_{25})$$

6. We get the following s-o-p expressions, where
   - the absorbed terms are without index
   - the remaining terms get an index j $(j=1,2,\dots)$

   to be identified for further calculations.

$c_{1a} \times c_{2a}$

$$= c_{14}\,(c_{21} \cup c_{26}) \cup c_{15}\,(c_{21} \cup c_{26}) \cup c_{18}\,(c_{21} \cup c_{22})$$

$$= 2\cdot3\cdot6 + \underset{(1)}{2\cdot3} + \underset{(4)}{3\cdot6\cdot14} + 2\cdot3\cdot6\cdot14 + \underset{(8)}{1\cdot3\cdot6} + 1\cdot2\cdot3\cdot6$$

$c_{1a} \times c_{2b}$

$$= c_{14}\,(c_{23} \cup c_{24}) \cup c_{15}\,(c_{23} \cup c_{24}) \cup c_{18}\,(c_{23} \cup c_{24})$$

$$= 2\cdot3\cdot1\cdot6\cdot14\cdot15 + 2\cdot3\cdot7\cdot8\cdot13 + 1\cdot3\cdot6\cdot14\cdot15 + 3\cdot6\cdot7\cdot8\cdot13\cdot14$$
$$+ \underset{(2)}{1\cdot6\cdot14\cdot15} + \underset{(9)}{1\cdot6\cdot7\cdot8\cdot13}$$

$c_{1a} \times c_{2c}$

$$= c_{14}\,(c_{22} \cup c_{25}) \cup c_{15}\,(c_{22} \cup c_{25}) \cup c_{18}\,(c_{22} \cup c_{25})$$

$$= 2\cdot3\cdot4\cdot12 + 2\cdot3\cdot7\cdot12\cdot15 + 3\cdot4\cdot6\cdot12\cdot14 + 3\cdot6\cdot7\cdot12\cdot14\cdot15$$
$$+ \underset{(11)}{1\cdot4\cdot6\cdot12} + \underset{(12)}{1\cdot6\cdot7\cdot12\cdot15}$$

$c_{1b} \times c_{2a}$

$$= c_{11}\,(c_{21} \cup c_{26}) \cup c_{12}\,(c_{21} \cup c_{26}) \cup c_{17}\,(c_{21} \cup c_{26})$$

$$= 2\cdot5\cdot3\cdot6 + 2\cdot5\cdot3 + \underset{(3)}{3\cdot6\cdot1o} + 2\cdot3\cdot6\cdot10 + \underset{(18)}{8\cdot9\cdot13\cdot3\cdot6} + 8\cdot9\cdot13\cdot2\cdot3$$

$c_{1b} \times c_{2b}$

$$= c_{11}\,(c_{23} \cup c_{24}) \cup c_{12}\,(c_{23} \cup c_{24}) \cup c_{17}\,(c_{23} \cup c_{24})$$

$$= 2\cdot5\cdot1\cdot6\cdot14\cdot15 + 2\cdot5\cdot7\cdot8\cdot13 + 8\cdot9\cdot13\cdot1\cdot6\cdot14\cdot15 + 3\cdot6\cdot10\cdot1\cdot6\cdot14\cdot15$$
$$+ 3\cdot6\cdot10\cdot7\cdot8\cdot13 + 8\cdot9\cdot13\cdot7$$

$$= 1\cdot2\cdot5\cdot6\cdot14\cdot15 + \underset{(5)}{2\cdot5\cdot7\cdot8\cdot13} + 1\cdot6\cdot8\cdot9\cdot13\cdot14\cdot15 + 1\cdot3\cdot6\cdot10\cdot14\cdot15$$

$$+ 3\cdot6\cdot7\cdot8\cdot10\cdot13 + \underset{(19)}{7\cdot8\cdot9\cdot13}$$

$c_{1b} \times c_{2c}$

$$= c_{11}\,(c_{22} \cup c_{25}) \cup c_{12}\,(c_{22} \cup c_{25}) \cup c_{17}\,(c_{22} \cup c_{25})$$

$$= \underset{(7)}{2\cdot4\cdot5\cdot12} + \underset{(6)}{2\cdot5\cdot7\cdot12\cdot15} + 2\cdot3\cdot4\cdot6\cdot10 + 3\cdot6\cdot7\cdot10\cdot12\cdot15$$

$$+ \underset{(20)}{4\cdot8\cdot9\cdot12\cdot13} + 7\cdot8\cdot9\cdot12\cdot13\cdot15$$

$c_{1c} \times c_{2a}$

$$= c_{13}\,(c_{21} \cup c_{26}) \cup c_{16}\,(c_{21} \cup c_{26})$$

$$= 3\cdot6\cdot10\cdot16 + 2\cdot3\cdot10\cdot16 + 5\cdot10\cdot11\cdot3\cdot6 + 15\cdot10\cdot11\cdot2\cdot3$$

$$= 3\cdot6\cdot10\cdot16 + 3\cdot5\cdot6\cdot10\cdot11$$

$c_{1c} \times c_{2b}$

$$= c_{13}\,(c_{23} \cup c_{24}) \cup c_{16}\,(c_{23} \cup c_{24})$$

$$= 10\cdot16\cdot1\cdot6\cdot14\cdot15 + 10\cdot16\cdot7\cdot8\cdot13 + 5\cdot10\cdot11\cdot1\cdot6\cdot14\cdot15$$

$$+ 5\cdot10\cdot11\cdot7\cdot8\cdot13$$

$$= 1\cdot6\cdot10\cdot14\cdot15\cdot16 + \underset{(10)}{7\cdot8\cdot10\cdot13\cdot16} + 1\cdot5\cdot6\cdot10\cdot11\cdot14\cdot15$$

$$+ \underset{(15)}{5\cdot7\cdot8\cdot10\cdot11\cdot13}$$

$c_{1c} \times c_{2c}$

$$= c_{13}\,(c_{22} \cup c_{25}) \cup c_{16}\,(c_{22} \cup c_{25})$$

$$= 10\cdot16\cdot4\cdot12 + 10\cdot16\cdot7\cdot12\cdot15 + 5\cdot10\cdot11\cdot4\cdot12 + 5\cdot10\cdot11\cdot7\cdot12\cdot15$$

$$= \underset{(17)}{4\cdot10\cdot12\cdot16} + \underset{(16)}{7\cdot10\cdot12\cdot15\cdot16} + \underset{(14)}{4\cdot5\cdot10\cdot11\cdot12} + \underset{(13)}{5\cdot7\cdot10\cdot11\cdot12\cdot15}$$

These results can be obtained by an algorithm (see /11/). We will here simply list the s-o-p-expression for Φ_{E1} Φ_{E2}

which is a unique (irredundant) cover by prime implicants (see also sect. 1.5, minimal cuts).

Table of prime implicants

Index j	Term p_j	Index j	Term p_j
1	$2\cdot 3$	11	$1\cdot 4\cdot 6\cdot 12$
2	$1\cdot 6\cdot 14\cdot 15$	12	$1\cdot 6\cdot 7\cdot 12\cdot 15$
3	$3\cdot 6\cdot 10$	13	$5\cdot 7\cdot 10\cdot 11\cdot 12\cdot 15$
4	$3\cdot 6\cdot 14$	14	$4\cdot 5\cdot 10\cdot 11\cdot 12$
5	$2\cdot 5\cdot 7\cdot 8\cdot 13$	15	$5\cdot 7\cdot 8\cdot 10\cdot 11\cdot 13$
6	$2\cdot 5\cdot 7\cdot 12\cdot 15$	16	$5\cdot 7\cdot 10\cdot 11\cdot 12\cdot 15$
7	$2\cdot 4\cdot 5\cdot 12$	17	$4\cdot 5\cdot 10\cdot 11\cdot 12$
8	$1\cdot 3\cdot 6$	18	$3\cdot 6\cdot 8\cdot 9\cdot 13$
9	$1\cdot 6\cdot 7\cdot 8\cdot 13$	19	$7\cdot 8\cdot 9\cdot 13$
10	$7\cdot 8\cdot 10\cdot 13\cdot 16$	20	$4\cdot 8\cdot 9\cdot 12\cdot 15$

All terms for the s-o-p-expression of Φ

$$\Phi = \Phi_{E1} \cdot \Phi_{E2}$$

$$= \sum_{j=1}^{20} p_j$$

1.8 Cubical Representation of Boolean Functions

We defined a switching function as a correspondence which assigns for each of the 2^n combinations of $x_1, x_2, \cdots, x_n$ one value of $\{0,1\}$. E.g. for a switching function

$$f\,(\overline{x}_1, \overline{x}_2, x_3) \;=\; x_1\, x_2 + x_2\, x_3$$

for each of $2^3 = 8$ combinations of x_1, x_2, x_3 a value of $\{0,1\}$ is assigned (Fig. 10).

x_1	x_2	x_3	$f\,(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Fig. 10 a Truth Table

x_3 \ $x_1\,x_2$	0 0	0 1	1 1	1 0
0	0	0	1	0
1	0	1	1	0

Fig. 10 b Map

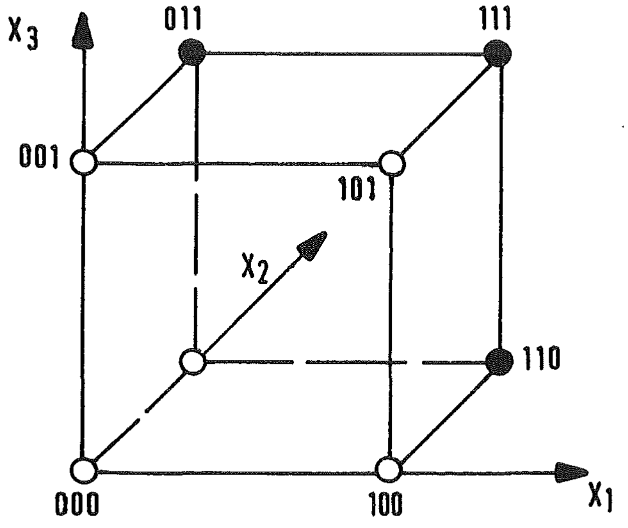


Fig. 10 c Cubical Representation

p_1	=	1 1 -
p_2	=	- 1 1

Prime Implicants

Thus the set of all 2^n combinations of

$$p_i = x_1, x_2, \cdots, x_n$$

with the corresponding values (1,0) is called a <u>cubical representation</u> of f $(x_1, x_2, \cdots, x_n)$. E.g. the set all all 2^3 combinations of

$$x_1, x_2, x_3$$

with the corresponding values (1,0)(see Fig. 10) is called a cubical representation of

$$f\ (x_1, x_2, x_3) = x_1\ x_2 + x_2\ x_3$$

(see also Fig. 10).

Each subset of the 2^n combinations generated by fixing some variables, while others take values (1,0) is called a <u>subcube.</u>

<u>Examples</u>

1. We obtain a subcube of Fig. 10c, fixing $x_3 = 0$, while x_1, x_2 may take values 1,0.

2. We obtain prime implicants of f (x_1, x_2, x_3) fixing $x_1 = 1$, $x_2 = 1$, while x_3 may take values 1,0 $(p_1 = x_1\ x_2)$, and fixing $x_2 = 1$, $x_3 = 1$ while x_1 may take values 1,0 $(p_2 = x_2\ x_3)$.

3. We obtain minterms fixing $x_1\ x_2\ x_3$, also called a 0-dimensional subcube.

Adjacent Subcubes

Let p_1 be a prime implicant which is represented as subcube. Then each subcube which differs in exactly one variable (say the k^{th} variable) from p_i will be called the adjacent subcube p^x_{ik} /1/.

Of course, this concept can be generalized. But this will be sufficient for our purposes.

E x a m p l e

		1	2	3	
$p_1 = x_1 x_2$		1	1	-	(prime implicant represented as subcube)
Adjacent subcubes					
p^x_{1k}	$k = 1$	0	1	-	
	$k = 2$	1	0	-	
$p_2 = x_2 x_3$		-	1	1	
Adjacent subcubes					
p^x_{2k}	$k = 1$	-	0	1	
	$k = 2$	-	1	0	

Note:
If a prime implicant p_i consists of l literals, the number of adjacent subcubes p^x_{ik} is l $(k = 1,2,\cdots,l)$.

2. Introduction to Failure Diagnosis

2.1 Types of Faults

2.2 Basic Concepts of Failure Diagnosis

2.3 Boolean Difference and Tests

2.4 Interpretation of Redundancy

2.1 Types of Faults

We assume Combinational Circuits. There are various types of failures /12/:

- permanent faults
- intermittent faults.

We only deal with permanent faults. If they are present, they will remain (until a repair is done). The permanent faults fall into two classes:

1. Classical faults, i.e.
   - stuck at zero (s-a-o)
   - stuck at one (s-a-1)

where a failed item behaves as if it had always the value 0 or 1.

Example:

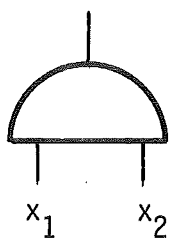


Fig. 11a

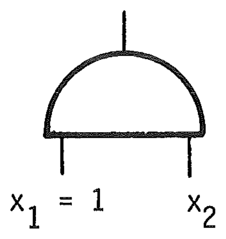


Fig. 11b

The circuit of Fig. 11a has for x_1 a s-a-1-fault (Fig. 11b).

Note

It will be our purpose to model all faults as logical faults. Thus the problem of failure diagnosis becomes a logical problem which is usually independent of the technology used. The same fault model is applicable to various technologies /12/.

2. Non-classical faults
   - e.g. Bridge faults
   - and others.

Example:

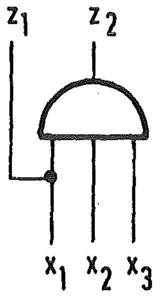


Fig. 12a

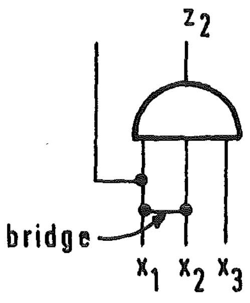


Fig. 12b

It can be seen that the bridge-fault leads to Boolean expressions for

$$z_1, z_2$$

which differ from Fig. 12a. We will not deal explicitly with these faults (/12/). Note that non-classical faults have no evident relation to systems represented by fault trees.

2.2. Basic Concepts of Failure Diagnosis

Now some basic notions for failure diagnosis of combinational circuits will be given /1/, /12/. Let C be a combinational circuit which realizes the function

$$f = f\ (x_1, x_2, \dots, x_n)$$

Let α be an arbitrary fault in the combinational circuit, where a number of variables change the output f to f_α .

Def. If $f_\alpha \neq f$ for at least one input $x_1, x_2, \dots, x_n$, we call the fault α detectable.

If $f_\alpha = f$ for all inputs $x_1, x_2, \dots, x_n$, we call the fault α undetectable.

Def. If for two faults α, α', and for all inputs,

$$f_\alpha = f_{\alpha'},$$

we call these faults functionally equivalent. There are in general equivalence classes of faults. A fault can be identified up to an equivalence class.

Example:

Let C be the following combinational circuit:

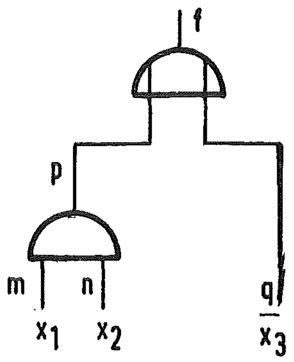


Fig. 13 Combinational Circuit

which realizes the function

$$f = x_1 x_2 + \overline{x}_3$$

Let f denote the fault free output and let f_α denote the output of this circuit in presence of fault α.

Denote by

$\alpha = m_0$ the fault of wire m, s-a-0

$\alpha = m_1$ the fault of wire m, s-a-1,

similarly n_i, p_i, q_i (i = 0,1).

The truth-table for this circuit is shown in Fig. 14. Here all possible single faults α are indicated.

Input			f	f_α							
x_1	x_2	x_3		f_{m_0}	f_{n_0}	f_{p_0}	f_{q_0}	f_{m_1}	f_{n_1}	f_{p_1}	f_{q_1}
0	0	0	1	1	1	1	0	1	1	1	1
0	0	1	0	0	0	0	0	0	0	1	1
0	1	0	1	1	1	1	0	1	1	1	1
0	1	1	0	0	0	0	0	1	0	1	1
1	0	0	1	1	1	1	0	1	1	1	1
1	0	1	0	0	0	0	0	0	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	1	1	1	1	1

Fig. 14

We observe (fig. 15) that

- collumns f_{m_0}, f_{n_0}, f_{p_0} are identical for all possible inputs, i.e. they are equivalent (cannot be distinguished), similarly f_{p_1}, f_{q_1}, are equivalent,
- there is no fault which is undetectable.

It is possible to simplify the fault table, which will be done below, but which is of little practical value.

Def. A test for fault α is an input $(x_1, x_2, \cdots, x_n)$ if in response to this input the output f_α is different from f.

Example:

Input			Possible faults				
x_1	x_2	x_3	$\{m_0, n_0, p_0\}$	q_0	m_1	n_1	$\{p_1, q_1\}$
0	0	0		1			
0	0	1					1
0	1	0		1			
0	1	1			1		1
1	0	0		1			
1	0	1				1	1
1	1	0					
1	1	1	1				

Fig. 15 Simplified fault table

We note:

- the only test for $\{m_o, n_o, p_o\}$ is 111;
- q_o can be tested by 000 or 010 or 100;
- m_1 can be tested by 011, provided there is no response for 001 and 101;
- n_1 can be tested by 101, provided there is no response for 001,011;
- $\{p_1, q_1\}$ can be tested by 001, 011, 101, provided there is a response for all three inputs.

Note: A fault table (Fig. 15) is a table in which there is a row for each possible test and a collumn for every fault. A "1" is entered at the intersection of the i-th row and the j-th collumn if the fault corresponding to the j-th collumn can be detected by the i-th test.

The problem of finding the minimal test set is closely related to the problem of finding a minimal cover of a Boolean function (by prime implicants). We will come back on a similar technique in section 5.

2.3 Boolean Difference and Tests

Assume a circuit C which realizes the oolean function

$$f\ (x_1, x_2, \cdots, x_n).$$

Let α be a fault in which input x_i is s-a-o. Then the function realized by this faulty circuit is

$$f_\alpha = f\ (x_1, x_2, \cdots, x_{i-1}, 0, x_{it1}, \cdots, x_n)$$

$$= f\ (o_i)$$

Similarly, if x_i is s-a-1, the function realized by the faulty circuit is

$$f_\alpha = f\ (x_1, x_2, \cdots, x_{i-1}, 1, x_{it1}, \cdots, x_n)$$

$$= f\ (1_i)$$

The Boolean difference method is an algebraic procedure to determine a complete set of tests to detect a given fault /1/.

<u>Def.</u> The Boolean difference of function $f\ (x_1, x_2, \cdots, x_n)$ with respect to its variable x_i is defined as

$$\frac{d\ f\ (\underline{x})}{d\ x_i} \begin{aligned} = {} & f\ (x_1,\ x_2,\ \cdots,\ x_{i-1},\ o,\ x_{i+1},\ \cdots,\ x_n) \\ & \oplus f\ (x_1,\ x_2,\ \cdots,\ x_{i-1},\ o,\ x_{i+1},\ \cdots,\ x_n) \end{aligned}$$

where $\oplus$ denotes the exclusive OR. It will be convenient to denote the Boolean difference as

$$\frac{d\ f\ (\underline{x})}{d\ x_i} = f\ (o_i) \oplus f\ (1_i)$$

<u>Rules:</u>

1. If $f\ (o_i) \oplus f\ (1_i) \equiv o$ for all variables, the fault related to x_i is undetectable (redundant).

2. We get all tests for s-a-o-faults if

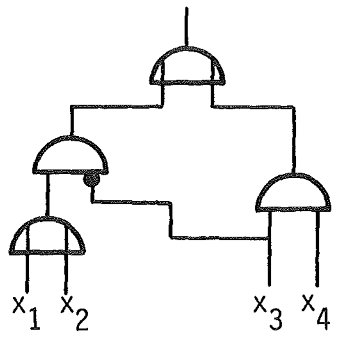
$$x_i \cdot \frac{d\ f\ (\underline{x})}{d\ x_i} = 1.$$

3. We get all tests for s-a-1-faults if

$$\bar{x}_i \cdot \frac{d\ f\ (\underline{x})}{d\ x_i} = 1.$$

I.e. if we have input combinations $\underline{x}$ which fulfil the conditions (2), (3), we have tests for the respective faults of x_i.

Example 1



$$f = (x_1 + x_2)\,\overline{x}_3 + x_3\,x_4$$

Fig. 16 Combinational circuit

We are interested in possible failures related to x_3. The Boolean difference with respect to x_3 is

$$\frac{d\,f\,(\underline{x})}{d\,x_3} = f\,(o_3) \oplus f\,(1_3)$$

$$= (x_1 + x_3) \oplus x_4 = \overline{x}_1\,\overline{x}_2\,x_4 + x_1\,\overline{x}_4 + x_2\,\overline{x}_4$$

For a s-a-o fault at x_3 we get with

$$x_3\,\frac{d\,f\,(\underline{x})}{d\,x_3} = \overline{x}_1\,\overline{x}_2\,x_3\,x_4 + x_1\,x_3\,\overline{x}_4 + x_2\,x_3\,\overline{x}_4 = 1$$

Thisexpression is equal to one if any of the product terms is equal to one. Thus we get as tests:

$$(x_1,x_2,x_3,x_4) = \{(o,o,1,1)\ ,\ (1,*,1,o),\ (*,1,1,o)\}$$

The DONT CARE-sign "*" tells that we are free to choose o or 1.

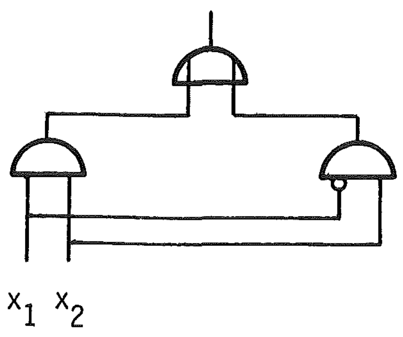
For a s-a-1 fault at x_3 we get with

$$\overline{x}_3\,\frac{d\,f\,(\underline{x})}{d\,x_3} = \overline{x}_1\,\overline{x}_2\,\overline{x}_3\,x_4 + x_1\,\overline{x}_3\,\overline{x}_4 + x_2\,\overline{x}_3\,\overline{x}_4 = 1,$$

as tests

$$(x_1\ x_2\ x_3\ x_4) = (o,\ o,\ o,\ 1),\ (1, *, o,\ o),\ (*,\ 1,\ o,\ o)$$

Example 2:



$f = x_1 x_2 + x_1 \bar{x}_2$

Fig. 17 Combinational Circuit

Is an error at input x_2 detectable?

$$\frac{d\, f\, (\underline{x})}{d\, x_2} = f\,(o_2) \oplus f\,(1_2)$$

$$= x_1 \cdot o \oplus x_1 \cdot \bar{1}$$

$$= o,$$

i.e. an error at input x_2 is not detectable.

Note:

Some interesting developments of the Boolean difference are:

- There are various rules which make the application for subsystems (subcircuits) easier.
- There is a generalization of Boolean difference for multiple faults.
- The Boolean difference is only for relatively small systems.

There are many methods for failure diagnosis available /1/, /12/.
We will deal with a few methods in sect. 5.2 and 5.3 of this report.

2.4 Interpretation of Redundancy

Sometimes, an interpretation of redundancy is desirable, which is not directly related to the detectability of failures.

Assume, we have a circuit which consists only of inputs, outputs and gates (AND, OR, NOT) and is acyclic (contains no directed circuits).

This type of combinational circuit is sometimes called 'wellformed' /2/ and will be considered here.

Definition:

Let N (Z) be a set of (wellformed) networks, which realize a given (Multioutput) combinational function

$$Z = (z_1, z_2, \cdots, z_m)$$

where

$$z_1 = f_1 (x_1, x_2, \cdots, x_{n1})$$

$$z_2 = f_2 (x_1, x_2, \cdots, x_{n2})$$

$$\vdots$$

$$z_m = f_m (x_1, x_2, \cdots, x_{n_m})$$

A network $N \in N (Z)$ is redundant if it is possible to remove lines and gates from N in such a way that the resulting network N' is in N (Z), and still realizes the same switching function.

A network which is not redundant will be called irredundant.

Note

A wellformed circuit can be defined recursively. We only mention one of its properties: A wellformed circuit is acyclic, i.e. it has no closed loop or feedback /1/. Also the fault trees (sect. 3.1) are wellformed circuits.

Examples:

1. $f(x_1, x_2) = x_1 x_2 + x_1 \overline{x}_2$ (N)

   Since

   $$x_1 x_2 + x_1 \overline{x}_2 = x_1 (x_2 + \overline{x}_2)$$
   $$= x_1$$

   we can delete lines and gates related to x_2. Only

   $f(x_1, x_2) = x_1$ (N')

   is needed. This is equivalent to saying that the circuit (N) is redundant.

2. A circuit, represented as a sum of prime implicants (without complements).

   $z = f(x_1, x_2, x_3)$

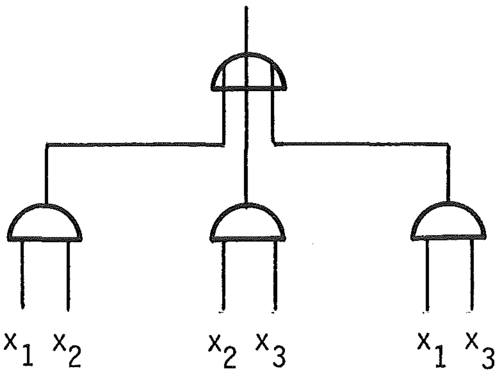


Fig. 18 Irredundant circuit

As can be seen in section 3.3 ('coherence'), no line or gate can be omitted, if the circuit z has to realize the same Boolean function. This circuit is irredundant.

3. F a u l t   T r e e s

3.1 Definition of Fault-Trees

3.2 Structure Function

3.3 Coherence of Systems and Minimal Cuts

3.4 A few Results on Coherent Structure Functions

3.1 Definition of Fault-Trees

We define a fault-tree and discuss a few properties of fault-trees, also indicating some relations to switching theory /13/.

Definition

A fault-tree is a finite directed graph without (directed) circuits, Each vertex may be in one of several states. For each vertex a function is given which specifies its state in terms of the states of its predecessors. The states of those vertices without predecessors are considered the independent variables of the fault-tree.

Some general properties of a fault-tree:

- The vertices without predecessors are the inputs to the fault-tree, representing the components. We are interested in the state of every other vertex, but in particular with the state of one vertex without successors, an output vertex which we identify with the state of the system as a whole. The graphical term 'vertex' here is roughly synonymous with 'item' and generally denotes any level in the system, whether a component, sub-system or the whole system.
- We specialize to only two states per vertex. This makes all of the functions Boolean functions. We call one of the two states 'functioning', 'false' or 0, and the other 'failed', 'true' or 1.
- Note, that this difinition of a two-state fault-tree is equivalent to a combinational network with one output.
- The no-circuit condition in the graph is equivalent to the condition that the current output of a switching circuit is entirely determined by current inputs, without memory of previous inputs or internal states.

3.2 Structure Function

We introduce the concept of structure function. It is of central importance for all problems of fault tree analysis /14 /, / 15 /, /16 /. It can be seen that it is closely related to the concept of switching function (see sect. 1.3).

We assume a system S, which has n components which can be in two states

- functioning
- failed.

Also the system S can be in two states, either functioning or failed. The components are the vertices without predecessors of our fault tree definition. The function which specifies the state of a vertex in terms of its predecessor is a Boolean function (AND, OR, NOT). The states of the top vertex can be given by a structure function.

Definition of Structure-Function

Let $x_1, x_2, \cdots x_n$ be Boolean variables which can assume the values 0,1, where

$$x_i = \begin{cases} o \text{ if component } i \text{ is functioning} \\ 1 \text{ if component } i \text{ is failed.} \end{cases}$$

The assumption that 1 corresponds to failure is used throughout this paper and is useful for fault tree analysis. The Boolean variable x_i indicates the state of component i, whereas the state vector

$$\underline{x} = (x_1, x_2, \cdots, x_n)$$

indicates the state of the system.
The Boolean function

$$\Phi (x_1, x_2, \cdots, x_n)$$

is called structure function and determines completely the state of the system S in terms of the state-vectors:

$$\Phi (x_1, x_2, \cdots, x_n) = \begin{cases} o \text{ if system S is functioning} \\ 1 \text{ if System S is failed.} \end{cases}$$

We note:

The structure function is related to the switching function as follows: They belong to two isomorphic algebraic systems. We call two algebraic systems isomorphic if they are identical up to the symbols used for operations and elements. Thus we can use all concepts and methods from switching algebra for fault tree analysis (and vice versa).

3.3 Coherence of Systems and Minimal Cuts

We introduced in sect. 1.1 the concept of completeness, especially referring to the set of operations

$$\{\text{AND, OR, NOT}\}.$$

This (and other complete sets) are usually used in switching algebra. In fault tree analysis we find quite frequently the set

$$\{\text{AND,OR}\},$$

which is not complete.(See examples in section 6.) We want to define coherence and show its relation to a simplified s-o-p representation, the minimal cut-representation. Note that failure diagnosis is not restricted to coherent systems (sect. 2 and 5) /14/, /15/, /16/.

Definition:

A system is called coherent if and only if

(a) a structure function exists which is nondecreasing in each variable, i.e.

$$\Phi(\underline{y}) \geq \Phi(\underline{x}) \quad \text{if}$$

$$\underline{y} \geq \underline{x} \quad \text{where}$$

$$y_i \geq x_i \quad (i = 1, \cdots, n),$$

(b) the relations hold

$$\Phi(\underline{o}) = o \quad \text{where } \underline{o} = (o, o, \cdots, o)$$

$$\Phi(\underline{1}) = 1 \quad \text{where } \underline{1} = (1, 1, \cdots, 1).$$

This means:

(a) If a system is functioning, then no transition of a component from failure to function can cause a system failure.

(b) If all components are functioning, the system is functioning. If all components are failed, then the system is failed.

Examples:

1. $\Phi(\underline{x}) = x_1 x_2 \vee x_2 x_3 \vee x, x_3$, representing a 2/3-system, is coherent.

2. $\Phi(\underline{x}) = x_1 \overline{x}_2 \vee \overline{x}_1 x_2$ representing an exclusive - OR - gate.

   (Fig. 1) is not coherent, since

   $(o, 1) \leq (1, 1)$ does not imply $\Phi(o, 1) \leq \Phi(1, 1)$.

3. Examples of coherent and noncoherent fault trees are given in sect.6.

Minimal Cut C_j

Let $M = \{K_i, K_2, \cdots, K_n\}$ be the set of components of a coherent system S. A subset V of M such that S is failed if all components belonging to V are failed and all componenets not belonging to V are not failed, is called a 'cut'. A cut is 'minimal' if no proper subsets exist which are also cuts. We call such a cut 'minimal cut' (C_j).

For each minimal cut it is possible, to find a combination of Boolean variables

$$\underline{x} = (x_1, x_2, \cdots, x_n).$$

Example:

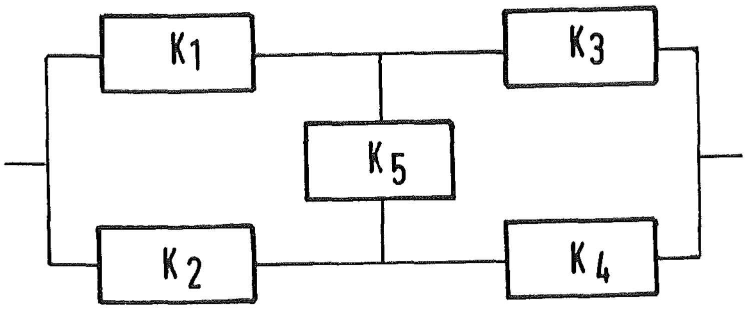
Network	minimal cuts
	$\{K_1, K_2\}$ $\{K_3, K_4\}$ $\{K_1, K_4, K_5\}$ $\{K_2, K_3, K_5\}$

Fig. 19 Network

Structure function

$$\Phi(1, 1, 0, 0, 0) = 1 \quad \text{(failed)}$$

but $$\Phi(0, 1, 0, 0, 0) = 0 \quad \text{(not failed)}$$

We write all components as K_i $(i = 1, 2, \ldots, n)$.
If a component K_i belongs to C_j we can use the notation $K_i \in C_j$.
For each minimal cut C_j we can use a structure function:

$$\alpha(C_j) = \bigwedge_{K_i \in C_j} x_i = \prod_{K_i \in C_j} x_i$$

The first expression is a conjunction of all K_i belonging to C_j.

The second expression is a multilinear form in x_i.

Example:

Let $C_1 = \{K_1, K_2\}$. Then,

$$\alpha(C_1) = \bigwedge_{K_i \in C_1} x_i = x_1 \wedge x_2 = x_1 x_2$$

Note that every min cut is a prime implicant without complements.
It is possible to express a coherent function using a sum of min cuts.

Example: For the network (Fig. 19) shown above, we get

$$(\underline{x}) = x_1 x_2 \vee x_3 x_4 \vee x_1 x_4 x_5 \vee x_2 x_3 x_5$$

or, as multi-linear-form:

$$\Phi(\underline{x}) = 1 - (1 - x_1 x_2)(1 - x_3 x_4)(1 - x_1 x_4 x_5)$$
$$\cdot (1 - x_2 x_3 x_4)$$

3.4 A few Results on Coherent Structure Functions

We mentioned in sect. 1.5 that every irredundant sum-of-products representation of a switching function is a union of prime implicants of this function. In section 3.2 we introduced the structure function which is isomorphic to the switching function. Moreover, we introduced the concept of coherence and the min cuts.

If the structure function is coherent, the representation by prime implicants greatly simplifies. We quote a theorem which leads to this simplification.

Theorem

A coherent structure function $\Phi(\underline{x})$ can be represented as a s-o-p

$$\Phi(\underline{x}) = \sum_{j=1}^{n} P_j$$

of prime implicants, where this representation is unique and can be written using the concept of min cuts

$$\Phi(\underline{x}) = \sum_{j=1}^{\ell} \prod_{K_1 \in C_j} x_i$$

where $K_i \in C_j$ are the components belonging to C_j, x_i the Boolean variables describing the states (functioning, failed) of the components /16, 17/.

Note, that there
- is only one (minimal) cover, and there
- are only essential prime implicants which may not be replaced by any other prime implicants.

This has the following consequences for the search for minimal cuts. The algorithm 3 (top-down-algorithm) or 4 (bottom-up-algorithm) leads to all min-cuts. Algorithms like 1,2 (using the complement) are not needed for this type of search. It may be also interesting to note that the problem of testing considerably simplifies if coherent structures are given. One of the simplifications will be evident in sections 4 and 5 (search for min-cuts instead of prime implicants for coherent structures).

4. Diagnosis Procedures

4.1 Diagnosis Procedure 'a'

4.2 Diagnosis Procedure 'b'

4. Some Diagnosis Procedures

Assume a system where for each relevant component a component failure is automatically detected. E.g. some systems of the Automated Laboratory for the WAK allow this type of failure detection /18, 19/.

The possible size of a fault table (dictionary and the use of Boolean differences (see sect. 2))is soon impractical. Thus, a method is needed which

- skips redundant information,
- decreases alarms which unnecessarily contribute to system unavailability
- may be used for realistic systems.

We discuss the following two types of tests:

(a) A test which leads to a prompt failure diagnosis for a failed system. This test is based on a structure function with minimal cuts.
The test aids to increase the availability of the system.

(b) A test which finds all states adjacent to system failure but only these. This test is based on a structure function with minimal cuts.
The test aids to increase the safety but the unavailability due to repair remains moderate.

Both tests can be used for systems which are not coherent as well (see sect. 5).

4.1 Diagnosis Procedure 'a'

1. Given a system S in fault tree representation or series-parallel representation with structure function Φ, where

$$\Phi = \sum_{j=1}^{\ell} p_j .$$

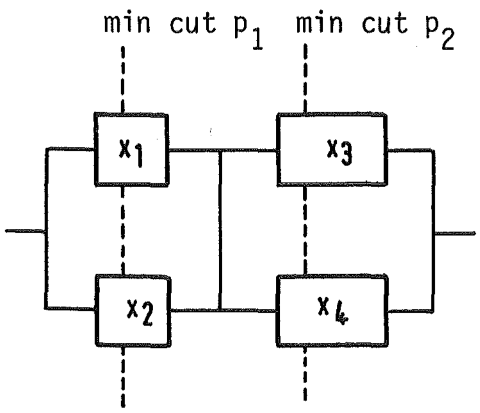
2. If a min cut p_j is equal to 1, there is system failure.

3. For all min cuts of Φ, test patterns (minterms) can be generated which uniquely determine whether a min cut is a cause for a system failure or not. This systematic account is called 'Diagnosis Procedure a' (Set of a-tests).

The relation to failure diagnosis concepts will be shown in sect. 5. It can be seen that no failure dictionary is needed. We give an example for 'Diagnosis Procedure a', (also called a-test).

Example:

① a-tests search for min cuts of f



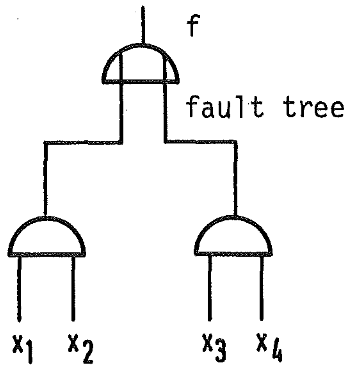


Fig. 20 System S_1

② Structure function $f = f(x_1, x_2, x_3, x_4)$

$$f = x_1 x_2 + x_3 x_4 \qquad x_i = \begin{matrix} o & \text{comp. i intact} \\ 1 & \text{comp. i failed} \end{matrix}$$

$$x_1 x_2 = o \qquad f = \begin{matrix} o & \text{system intact} \\ 1 & \text{system failed} \end{matrix}$$

$$x_3 x_4 = o$$

③ a-test

Minterms \ p	$x_1\ x_2\ x_3\ x_4$ 1 1 - -	$x_1\ x_2\ x_3\ x_4$ - - 1 1
1 1 o o	1	o
o o 1 1	o	1

$$x_1 x_2 = \begin{matrix} o \\ 1 \end{matrix} \text{ min cut } \{1,2\} \begin{matrix} \text{intact} \\ \text{failed} \end{matrix}$$

$$x_3 x_4 = \begin{matrix} o \\ 1 \end{matrix} \text{ min cut } \{3,4\} \begin{matrix} \text{intact} \\ \text{failed} \end{matrix}$$

By the a-test we can determine, whether min cuts lead to system failure or not. Every min cut which has value 0, is not a cause for system failure 1*). The min cut which has value 1 is the cause for system failure. A search for components is not needed. The entire cut needs repair.

1*) In some cases also a direct search for the responsible cut may be possible, simply searching for the cut which has value 1.

4.2 Diagnosis Procedure 'b'

1. Given a system S in fault tree representation or series parallel representation with structure function Φ, where

$$\Phi = \sum_{k=1} p_k .$$

2. If a min cut p_k is equal to 1, the system fails.

3. For all min cuts p_k adjacent subcubes p_{ik}^{x} can be found which refer to states of a coherent system where only one more component has to fail to cause a system failure.

4. Test patterns can be generated uniquely determining the states adjacent to system failure. This systematic account is called 'Diagnosis Procedure b' (set of b-tests).

The relation to failure diagnosis will be discussed in sect. 5. We give an example for Diagnosis Procedure a (also called b-test).

Example:

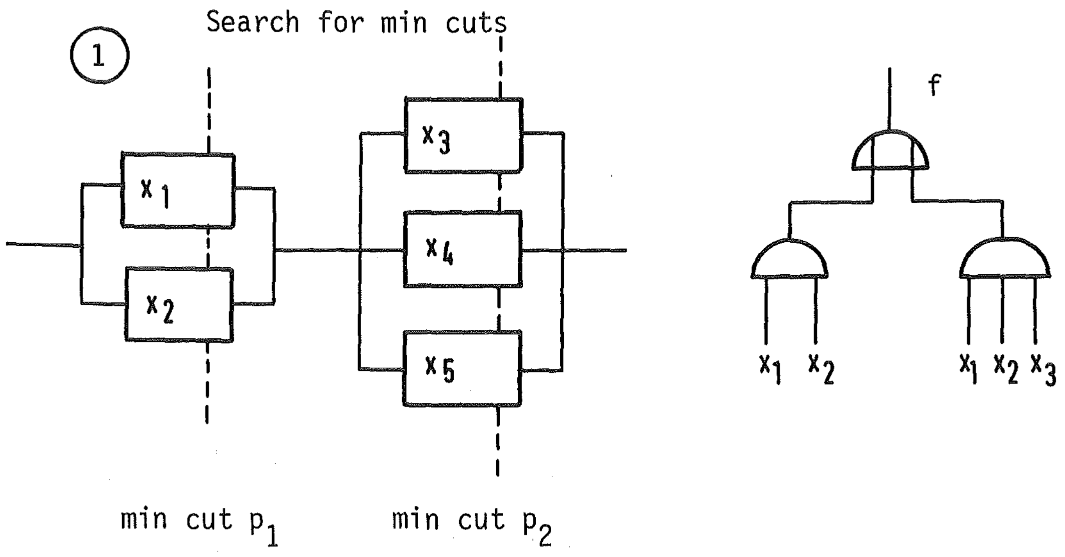


Fig. 21 System S_2

(2) Structure function f

$f = f(x_1, x_2, x_3, x_4, x_5)$

$f = x_1 x_2 + x_3 x_4 x_5$ (in min cuts)

(3) <u>b-test</u>

Let p_i be prime implicants (min cuts)

p^x_{ik} be adjacent subcubes to the p_i

$p_1 = x_1 x_2$		1 2 3 4 5	1 2 3 4 5
		1 1 - - -	Minterms
p^x_{1k}	k = 1	0 1 - - -	0 1 0 0 0
	2	1 0 - - -	1 0 0 0 0
$p_2 = x_3 x_4 x_5$		- - 1 1 1	
p^x_{2k}	k = 1	- - 0 1 1	0 0 0 1 1
	2	- - 1 0 1	0 0 1 0 1
	3	- - 1 1 0	0 0 1 1 0

We obtain all states of the system S_2 which are adjacent to system failure:

1.	component 1 failed:	p^x_{11} =	0 1 - - -
	component 2 failed:	p^x_{12} =	1 0 - - -
2.	component 4 and 5 failed:	p^x_{21} =	- - 0 1 1
	component 3 and 5 failed:	p^x_{22} =	- - 1 0 1
	component 3 and 4 failed:	p^x_{23} =	- - 1 1 0

By the b-test we can locate all states which are adjacent to system-failure. Then it is possible to prevent system failure replacing the failed components.

Clearly, all the techniques from a and b-Tests, also in relation with search for prime implicants (or min cub) can be applied for automatic diagnosis of systems. This will be shown in more detail in our next section.

5. Tests for Two Types of Faults

5.1 General Assumptions

5.2 Tests for s-a-0-Faults

5.3 Tests for s-a-1-faults

5.4 Examples for Tests

5.5 Existence of Tests

5.6 Relation to Diagnosis Procedures

Introduction

We discuss tests for two types of faults which occur in combinational networks:

- the stuck at one fault (s-a-1)
- the stuck at zero fault (s-a-0).

Other faults are not considered. Combinational networks are related to fault trees due to the isomorphism of switching function and structure function. We concentrate here on two tests which use prime implicants (or min cuts). They were developed in /1, 17/. These tests have been introduced on an informal basis in sect. 4 (Diagnosis Procedures a,b).

5.1 General Assumptions

We assume a two-level network (AND-OR-Type), or a network which can be transformed into an equivalent two-level network (i.e. without deletion of real failures and/or introduction of new failures). In Fig. 22, the AND-OR-type network is shown:

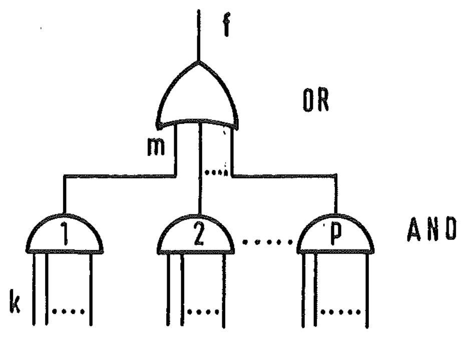


Fig. 22 AND-OR-network

We assume that this is an irredundant network which is equivalent to an irredundant sum of prime implicants. Thus, the switching function f can be written

$$f\ (x_1, x_2, \ldots, x_n) = \sum_{i=1}^{l} p_i$$

where p_i denotes the ith prime implicant, l is the number of prime implicants of the irredundant sum.

Each AND-gate is equivalent to one prime imlicant. Here we need an algorithm to search for prime implicants (see sect.1). If the system is coherent, a search for min cuts is sufficient (see sect. 1.7).
A circuit which consists of r wires may have as many as
2r distinct single faults (s-a-0, s-a-1), and 3^r-1 multiple faults (single, double, ..., r-tuple faults). This is due to the binomial theorem /1/:

$$3^r = (2+1)^r = \sum_{i=0}^{r} \binom{r}{i} \; 2^i \cdot 1^{r-i}\text{, where } 1^{r-i} = 1.$$

5.2 Tests for s-a-0 Faults

We discuss the tests for s-a-0 faults, which correspond to the 'Diagnosis Procedure a'.
A s-a-0 fault at any of the inputs of the j^{th} AND-gate causes the output of this gate to be s-a-0, regardless of the value of the remaining variables. Such a fault eliminates the corresponding prime implicant p_j from the function f:

$$f = \sum_{i=1}^{l} p_i$$

To check whether a given prime implicant p_i has completely vanished, it is sufficient to have one minterm a_j as input which is covered by that prime implicant p_j and by no other prime implicant /17/.

For all 'essential prime implicants' such a minterm exists, this is especially true for min cuts (unique representation). The requirement that a minterm a_j must be one that is covered by the prime implicant p_j and by no other prime implicant p_i $(i \neq j)$ is essential. We note, that a complete set of tests for s-a-0 faults for a s-o-p-network consists of n tests corresponding to the l prime implicants in f.

To test the j^{th} AND-gate for s-a-0 faults, it is necessary and sufficient to have as input one minterm a_j such that

$$a_j \varepsilon \, p_j \cdot \overline{\left(\sum_{\substack{i=1 \\ i \neq j}}^{l} p_i \right)} = p_j \cdot \prod_{\substack{i=1 \\ i \neq j}} \overline{p_i}$$

The systematic account of minterms a_j to test the AND-gates for s-a-0 is referred to as a set of a-tests. It can be shown that all single and multiple stuck-at-faults can be detected by this method.

We give an algorithm for generating the (minimal) a-tests.

We introduce the covering matrix E.

$$E = \begin{array}{c|cccc|} & p_1 & p_2 \cdots\cdots & p_j & p_n \\ m_1 & & & \vdots & \\ m_2 & & & \vdots & \\ \cdot & & & \vdots & \\ \cdot & & & \vdots & \\ \cdot & & & \vdots & \\ m_i & \cdots & \cdots & e_{ij} & \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ m_k & & & & \end{array}$$

The covering matrix shows for all minterms m_i if they are covered by prime implicants.

If m_i is covered by p_j, we have $e_{ij} = 1$,

m_i is not covered by p_j, we have $e_{ij} = 0$.

Algorithm 5

Step 1 Construct a covering matrix E whose collumn headings are p_j, and whose row headings are m_i.

Step 2 Delete all rows which contain two or more 1's.

Step 3 Is there a p_j which cannot be covered?

Step 4 Choose for every p_j in E one minterm a_j.
Thus we get the minterms

$$a_j = p_j \prod_{\substack{i=1 \\ i \neq j}} \overline{p_i}$$

$\square$

5.3 Tests for s-a-1 Faults

We discuss the tests for s-a-1 faults, which correspond to the "Diagnosis Procedure b". A s-a-1 fault at any of the inputs of the j^{th} AND-gate causes the prime implicant not to vanish. But the output of the gate becomes independent of the variable associated with a s-a-1 fault.

Example:

Let the input x_k of AND-gate $x_1x_2x_3$ s-a-1. This is for

$$k = 1 \qquad 1 \cdot x_2 \cdot x_3 = x_1x_2x_3 + \bar{x}_1x_2x_3$$

$$k = 2 \qquad x_1 \cdot 1 \cdot x_3 = x_1x_2x_3 + x_1\bar{x}_2x_3$$

$$k = 3 \qquad x_1x_2 \cdot 1 = x_1x_2x_3 + x_1x_2\bar{x}_3$$

To test the k^{th} input the j^{th} AND-gate for s-a-1-faults, it is necessary and sufficient to have as input one minterm b_{jk} such that

$$b_{jk} \varepsilon\, p^x_{jk} \left(\overline{\sum_{i=1}^{\ell} p_i} \right) = p^x_{jk} \prod_{i=1}^{\ell} \overline{p_i}$$

where

p^x_{jk} is a subcube adjacent to p_j (see sect. 1.8) and p_j is the j^{th} prime implicant.

The systematic account of minterms b_{jk} to test all AND - gates for s-a-1 faults is called a set of b-tests. It can be shown that all single and mutiple stuck-at-faults can be detected by this method.

Before giving the Algorithm a few remarks seem in order (see also Examples given below).

- Pairwise intersection: Assume a cubical representation (sect. 1.8.). For terms like 11--and-11-the pairwise intersection is 111-.
- Prime intersection: If intersecting with other terms leads to no further intersection, we have a prime intersection.
- Prime tests: The prime intersections are related to prime tests.
- Prime test chart: A chart with collumn headings p^x_{jk} and with row headings b_{jk} (prime tests) is called prime test chart.

With these remarks we can state our Algorithm.

Algorithm 6

Step 1 List all p^x_{jk} for all $j = 1,2, \ldots, l$ and $k = 1, 2, \ldots, r_j$ where l is the number of prime implicants and r_j the number of literals in the j^{th} prime implicant. Thus we get all adjacent subcubes.

Step 2 For every $p^x_{is} \supseteq p^x_{jt}$ delete p^x_{is} form list.

Step 3 Find all pairwise intersections of the terms that are now contained in the list. Whenever an intersection is nonempty and contains a minterm for which $f = 0$, checkmark the intersected terms. This step lists the minterms for which $f = 0$ which are contained in 2 or more adjacent subcubes.

Step 4 Repeat Step 3 until no new terms are generated. The terms generated in step 3 and those checkmarked in step 2 are called prime intersections. Steps 3 and 4 thus indicate those minterms which simultaneously test as many subcubes as possible.

Step 5 From the list of prime intersections construct a list of prime tests by selecting arbitrarily an input combination b_{jk} for which the value of the function is 0.

Step 6 Construct a prime test chart where the collumn headings are p^x_{jk} (found in step 2) and the row headings prime tests (found in step 5). A sign (x) is inserted at the intersection of any one row and collumn if the corresponding prime test is covered by p^x_{jk}. We get

$$b_{jk} \in p^x_{jk} \prod_{i=1}^{e} \bar{p}_i$$

Step 7 Select a set of prime tests that check each of the p^x_{jk} - terms, i.e. find a cover for the prime test chart. $\square$

5.4 Example for Tests (s-a-0 and s-a-1-faults)

Given the following network:

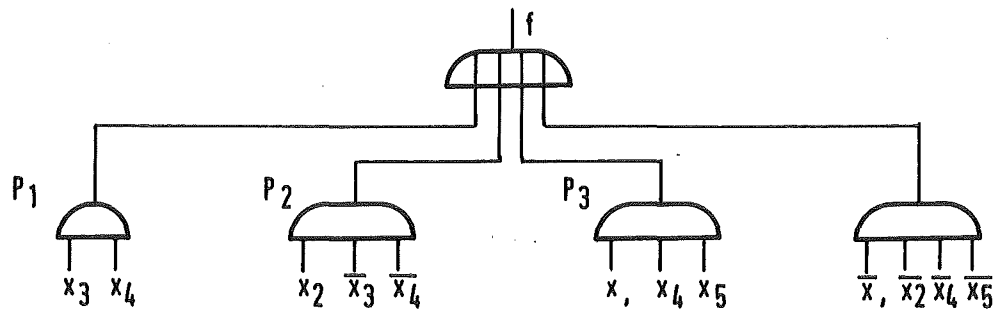

f
P_1
P_2
P_3
x_3 x_4
x_2 $\bar{x}_3$ $\bar{x}_4$
x_1 x_4 x_5
$\bar{x}_1$ $\bar{x}_2$ x_4 $\bar{x}_5$

Fig. 23

This can be represented as a sum of prime implicants:

$$f = \sum_{j=1}^{4} p_j = x_3x_4 + x_2\bar{x}_3\bar{x}_4 + x_1x_4x_5 + \bar{x}_1\bar{x}_2x_4\bar{x}_5$$

a-Test (s-a-0-faults)

It can be seen that each prime implicant covers at least one minterm which is not covered by any other prime implicant (see also Karnaugh-map, Fig. 24).

We write as covering matrix with headings
- p_j (collums)
- m_i (rows):

m_i p_j	p_1	p_2	p_3	p_4	
$x_1x_2x_3x_4x_5$	--11-	-100-	1--11	00-10	
01000	0	1	0	0	
11000	0	1	0	0	
00010	0	0	0	1	
00110	1	0	0	1	★
10110	1	0	0	0	
01110	1	0	0	0	
11110	1	0	0	0	
01001	0	1	0	0	
11001	0	1	0	0	
10011	0	0	1	0	
11011	0	0	1	0	
00111	1	0	0	0	
10111	1	0	1	0	★
01111	1	0	0	0	
11111	1	0	1	0	★

Covering Matrix

Step 1 Construction of covering matrix

Step 2 Delete all rows which contain more than one 1 (rows checkmarked by a ★).

Step 3 There is no p_j which cannot be detected by a minterm.

Note: These tests (a_j) are necessary and sufficient to test for all s-a-o-faults.

Step 4 We choose for every p_j one minterm a_j, e.g.

$$\{a\} = \{\underset{p_1}{11110},\ \underset{p_2}{11000},\ \underset{p_3}{10011},\ \underset{p_4}{00010}\}$$

Note: These tests (a_j) are necessary and sufficient to test for all s-a-0-faults.

Finally, we show a Karnaugh-map with prime implicants p_j and minterms a_j.

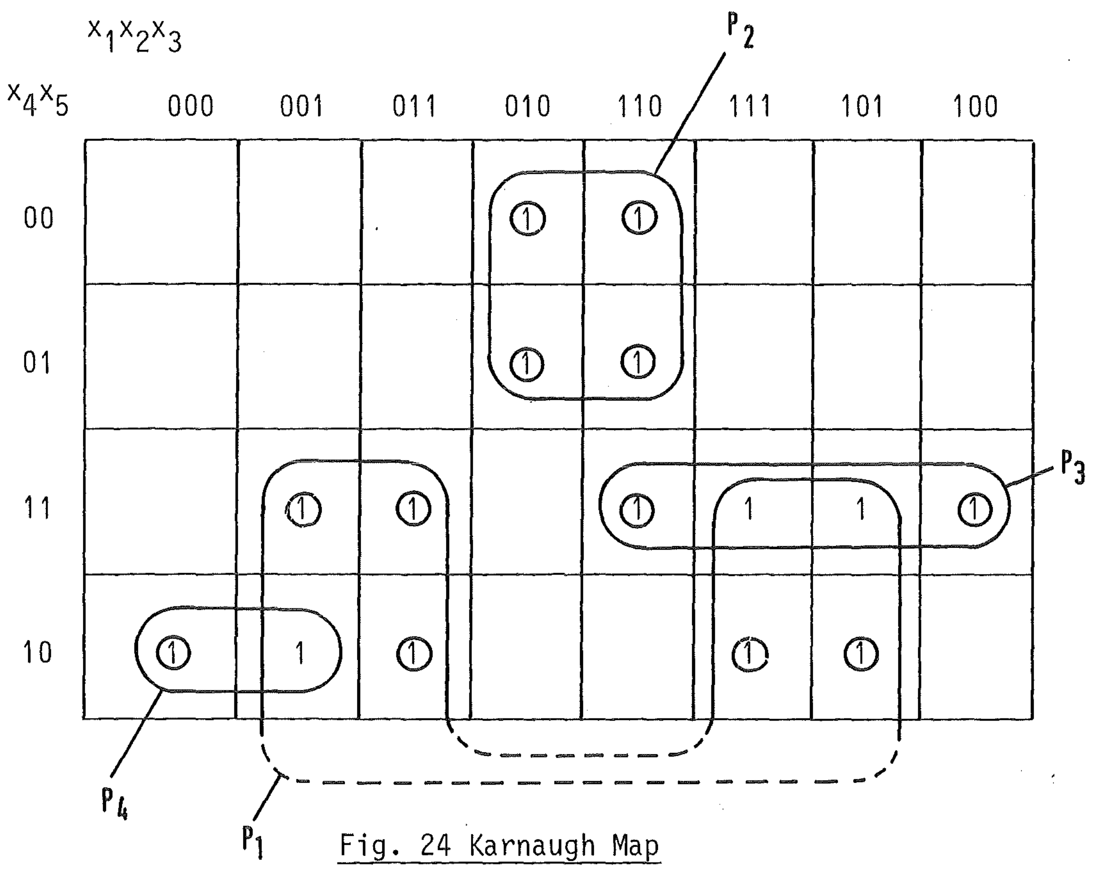


Fig. 24 Karnaugh Map

Note:
All the circled minterms belong to the minterms of f with

$$a_j \varepsilon p_j \cdot \prod_{\substack{i=1 \\ i \neq j}}^{e} \bar{p}_i$$

The minterms which are covered by more than one p_i have been deleted from the covering matrix E.

<u>b-Test</u> (s-a-1-faults)

<u>Step 1</u> From prime implicants p_j we find all adjacent subcubes p^x_{jk}.

p_1 = --11--	p^x_{11} = --10-
	p^x_{12} = --01-
p_2 = -100-	p^x_{21} = -000-
	p^x_{22} = -110-
	p^x_{23} = -101-
p_3 = 1--11	p^x_{31} = 0--11
	p^x_{32} = 1--01
	p^x_{33} = 1--10
p_4 = 00-10	p^x_{41} = 10-10
	p^x_{42} = 01-10
	p^x_{43} = 00-00
	p^x_{44} = 00-11

<u>Step 2</u> For every $p^x_{is} \supseteq p^x_{jt}$ delete p^x_{is}

We get p^x_{12} = --01- $\supset$ p^x_{22} = -110-

p^x_{11} = --10- $\supset$ p^x_{23} = -110-

p^x_{33} = 1--10 $\supset$ p^x_{41} = 10-10

p^x_{31} = 0--11 $\supset$ p^x_{44} = 00-11

Thus our new list is

p^x_{21}	-000-
p^x_{22}	-110-
p^x_{23}	-101-
p^x_{32}	1--01
p^x_{41}	10-10
p^x_{42}	01-10
p^x_{43}	00-00
p^x_{44}	00-11

<u>Step 3</u> We find pairwise intersections, e.g.

$p^x_{23} \cap p^x_{42}$ = -101- $\cap$ 01-10 = 01010.

We get:

01010, 11101, 10001, 00000

<u>Step 4</u> The prime intersections (where intersection leads to no further terms) are

10-10, 00-11
01010, 11101, 10001, 00000

<u>Step 5</u> To find a test from the intersection 00-11, note that this intersection covers two minterms

00011 and 00111

since 00111 ε --11- = p_1, only
00011 is admitted as a test.

We get as prime tests (minterms)

00011, 10010
01010, 11101, 10001, 00000

Step 6 The prime test chart is given next:

b_{jk} \ P^x_{jk}	-000-	-110-	-101-	1--01	10-10	01-10	00-00	00-11
00011								x
10010					x			
01010			x			x		
11101		x		x				
10001	x			x				
00000	x						x	

Note:
These tests (b_{jk}) are necessary and sufficient to test for all s-a-1-faults. We give no representation with Karaugh-map here.

The method of covering a prime test chart is similar to the covering of a fault table. But almost always, the size of a prime test chart is small compared with the corresponding fault table (see sect. 2.2).

5.5 Exixtence of Tests

Theorem: The set T of a-tests and b-tests detects all multiple faults in the two-level AND-OR-network, where all

a-tests are of the type $a_j \in p_j \cdot \prod_{\substack{i=1 \\ i\neq j}}^{\ell} \bar{p}_i$

and all

b-tests are of the type $b_{jk} \in p^x_{jk} \cdot \prod_{i=1}^{\ell} \bar{p}_i$.

Proof: We consider only the inputs x_i. If any s-a-0 or s-a-1 occurs in one of the inputs, it will be detected by the tests T. If any input is s-a-1, its effect is to add a subcube p^x_{jk} to the switching function. This subcube can only be deleted (i.e. the subcube will be with an undetectable fault s-a-1) if a s-a-0-faults on an input to the same AND-gate occurs.
This s-a-0 fault cannot be "masked" by another s-a-1 fault at the gate:

From $x_1.x_2..1_{i_1}..0_{i_2}..x_{nj}$ we get the vanishing of the prime implicant, therefore it will be detected by an a-test.

A s-a-0 at an input to an AND gate causes the prime implicant p_j to vanish. The p_j is tested by a single a test. If, however, this a test (minterm) is included at the same time in an adjacent subcube added to the switching function as a result of some s-a-1 fault, it will not detect the "vanished" prime implicant. The s-a-1, however, will be detected by the b-tests.

In all other situations the a test will detect all s-a-0 faults. The a-tests and b-tests together detect all multiple faults, but not necessarily a or b-tests alone. $\square$

This proof has been presented in /1/. Here the proof has been simplified to some extent.

5.6 Relation to Diagnosis Procedures

To apply our concepts correctly to Diagnosis Procedures (introduced in sect. 4) some relations will be outlined:

There is a close correspondence between

1. a-Tests (for s-a-0 faults) and a-Diagnosis Procedures (for failure diagnosis of systems represented by fault trees),

2. b-Tests (for s-a-1 faults) and b-Diagnosis Procedures (for diagnosis of subcubes adjacent to system failure).

Clearly, all the techniques from a and b-Tests, also in relation with search for prime implicants (or min cubes) can be applied for automatic diagnosis of systems. This will be shown in more detail in our next section.

6. Examples with Various Fault Trees

6.1 Subsystem of Automated Laboratory

6.2 Standby System with Motor

6.3 Failure of Residual Heat Removal System

6.4 Nitric Acid Cooler

6.5 An illustrative Fault Tree

6.1 Subsystem of Automated Laboratory

Here we regard the photometer and conductivity measurements, which have been discussed in more detail in /18/, as a first example (Fig. 25).

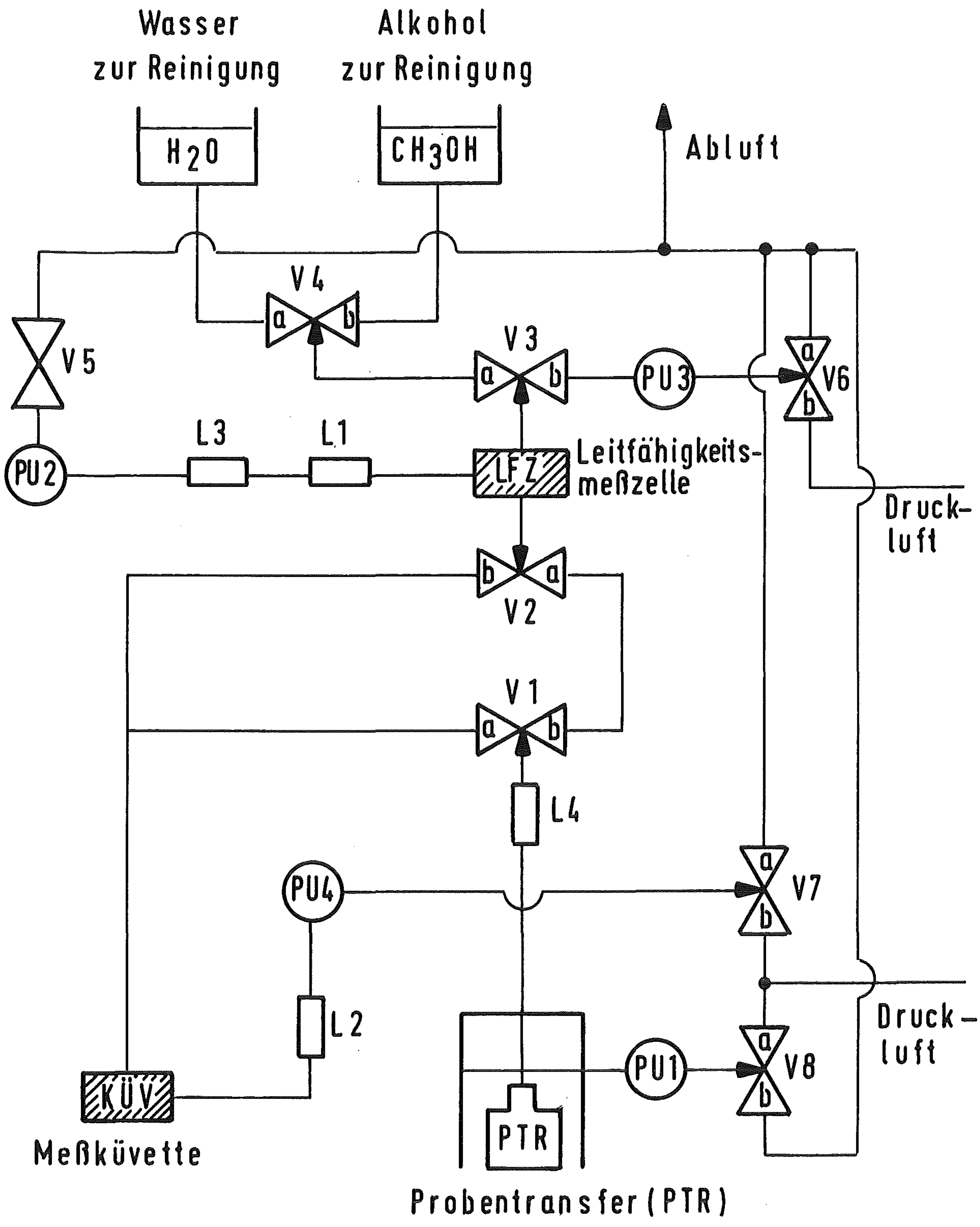


Fig. 25 Vereinfachtes Apparateschema (Schematic diagram of automated photometry and conductimetry system /18/).

In a schematic diagram this device is shown. Then a subtree leading to the event "Error in a photometer measurement" is show. From the related structure function we get

- a-tests and
- b-tests.

Component failures (Inputs), Fig. 26.

V1b, V2B, V3a as well as PU1 (full), V8a, L4 indicate failures in the components of the device. Note that for the analysis step No. 5 (cuvette filled) (see /18/, /19/) two min cuts may lead to a measurement error. Note that this event only reduces availability (not the safety) of this device. A fast diagnose is desirable to reduce unavailability.

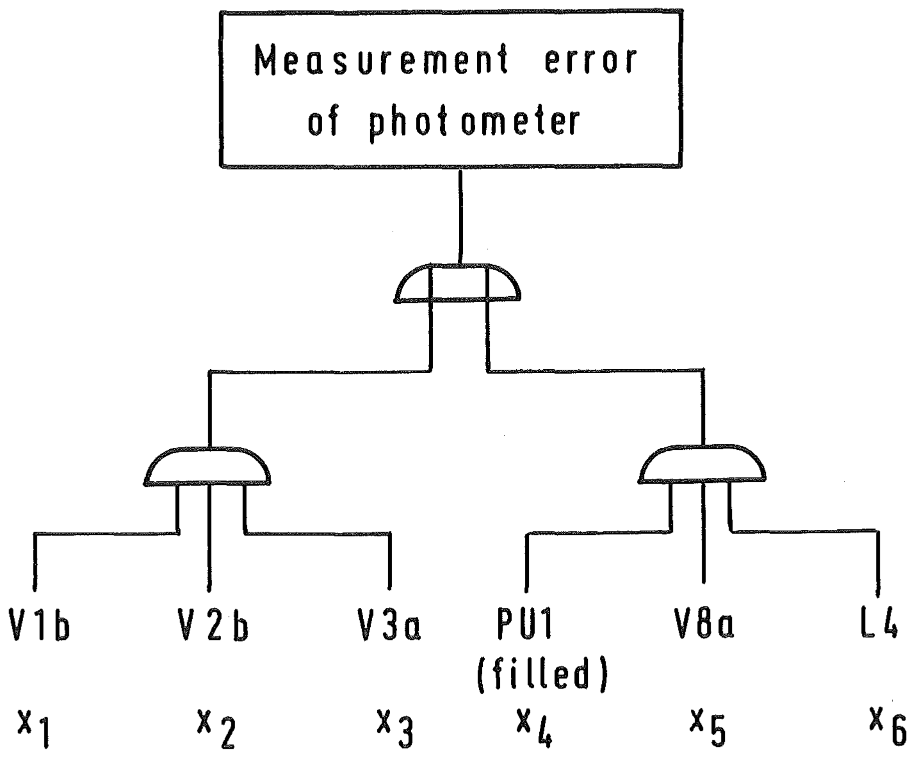


Fig. 26 Fault Tree

The structure function is:

$$\Phi = x_1x_2x_3 + x_4x_5x_6$$

We get as a-tests:

a_j \ p_j	123456 111---	123456 ---111
111000	1	0
000111	0	1

If two min cuts are possible causes of measurement error, we can exactly locate the failed component.

We get as b-tests:

$p_1 = x_1x_2x_3$		123456 111---		123456
p^x_{1k}	k=1	011---	b_{1k}	011000
	2	101---		101000
	3	110---		110000
$p_2 = x_4x_5x_6$		---111		
p^x_{2k}	k=1	---011	b_{2k}	000011
	2	---101		000101
	3	---110		000110

Thus we can detect all states which are adjacent to system failure. This is still much better than stop the device for any single failure, which considerably decreases unavailability. Moreover, this leads to a systematic search for all states adjacent to system failures in the whole operation of the device.

Note: This test set can be used for the whole photometry and conductivity measurement subsystem (see also /18/).

Efficiency: For n = 6 inputs we have

$3^n - 1$ multiple faults (including single faults), i.e.
$3^6 - 1 = 7.28 \cdot 10^2$

All are automatically contained in the Lists for a-tests and b-test.

6.2 A Standby System with Motor

This system is reproduced in the literature /20/. It has been used for fault tree analysis.

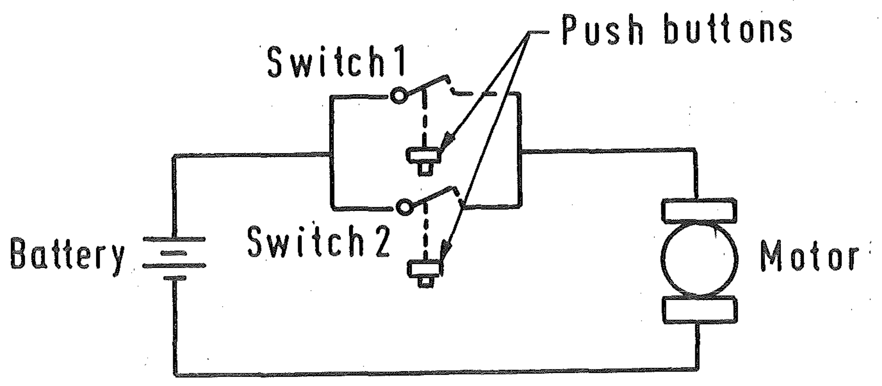


Fig. 27 Standby system

We describe this system shortly: Assume, the system is a standby system that is tested once every month. It consists of a battery, two switches in parallel, and a motor. To start the motor, two push buttons are pressed to close the two switch contacts 1 and 2. To stop the motor at the end of test, two push buttons are depressed. Periodically, say every six months, the operator must recharge the battery and perform routine maintenance on the motor.

We have the following fault tree which describes the failure of the motor to start on request.

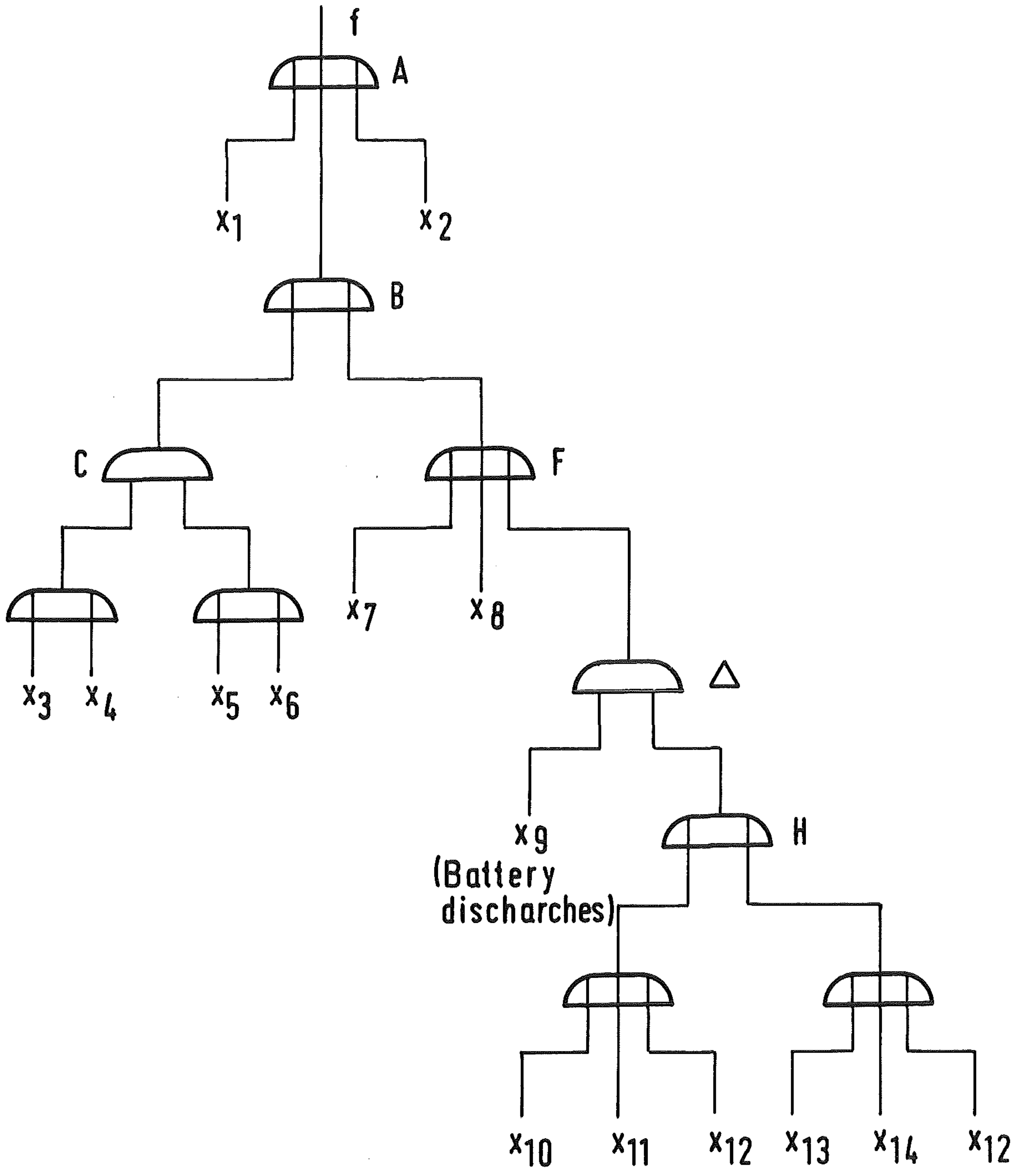


Fig. 28 Fault Tree

Next we give the structure function.
By a top-down algorithm we find the min cuts.

$$
\begin{aligned}
f &= x_1 + B + x_2 \\
&= x_1 + C + F + x_2 \\
&= x_1 + D \cdot E + x_7 + G + x_2 \\
&= x_1 + D \cdot E + x_7 + x_9 \cdot H + x_8 + x_2 \\
&= x_1 + x_2 + x_7 + x_8 + D \cdot E + x_9 \cdot H \\
&= x_1 + x_2 + x_7 + x_8 \\
&\quad + x_3 \cdot x_5 + x_3 \cdot x_6 + x_4 \cdot x_5 + x_4 \cdot x_6 \\
&\quad + x_9 \cdot x_{10} + x_9 \cdot x_{11} + x_9 \cdot x_{12} \\
&\quad + x_9 \cdot x_{13} + x_9 \cdot x_{14}
\end{aligned}
$$

We give a list of the min cuts, also describing the related failure combinations.

P_j	Min Cut Set	Description of failure combination
1	{1}	Motor fails to start
2	{2}	Inadequate maintenance of motor
3	{7}	Dead battery (primary failure)
4	{8}	Operator fails to recharge battery
5	{3,5}	Switch 1 contacts fail to close Switch 2 contacts fail to close
6	{3,6}	Switch 1 contacts fail to close Secondary failure of switch 2
7	{4,5}	Secondary failure of switch 1 Switch 2 contacts fail to close
8	{4,6}	Secondary failure of switch 1 Secondary failure of switch 2
9	{9,10}	Battery operates sufficiently long to discharge Secondary failure of switch 1
10	{9,11}	Battery operates sufficiently long to discharge Switch 1 contacts fail to open
11	{9,12}	Battery operates sufficiently long to discharge Operator fails to depress push button
12	{9,13}	Battery operates sufficiently long to discharge Switch 2 contacts fail to open
13	{9,14}	Battery operates sufficiently long to discharge Secondary failure of switch 2

List with failure combinations

a-Test

Prime Implicants

p_j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	-	-	-	-	-	-	-	-	-	-	-	-	-
2	-	1	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	1	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	1	-	-	-	-	-	-
5	-	-	1	-	1	-	-	-	-	-	-	-	-	-
6	-	-	1	-	-	1	-	-	-	-	-	-	-	-
7	-	-	-	1	1	-	-	-	-	-	-	-	-	-
8	-	-	-	1	-	1	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	1	1	-	-	-	-
10	-	-	-	-	-	-	-	-	1	-	1	-	-	-
11	-	-	-	-	-	-	-	-	1	-	-	1	-	-
12	-	-	-	-	-	-	-	-	1	-	-	-	1	-
13	-	-	-	-	-	-	-	-	1	-	-	-	-	1

Minterms	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	single failures
3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
5	0	0	1	0	1	0	0	0	0	0	0	0	0	0	
6	0	0	1	0	0	1	0	0	0	0	0	0	0	0	switches
7	0	0	0	1	1	0	0	0	0	0	0	0	0	0	fail to close
8	0	0	0	1	0	1	0	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	1	1	0	0	0	0	
10	0	0	0	0	0	0	0	0	1	0	1	0	0	0	
11	0	0	0	0	0	0	0	0	1	0	0	1	0	0	battery
12	0	0	0	0	0	0	0	0	1	0	0	0	1	0	discharge
13	0	0	0	0	0	0	0	0	1	0	0	0	0	1	
			switch 1		2										

The cuts p_j, causing the defect can be precisely located.

b-Test

p_1, p_2, p_3, p_4 are single Failures: b-test not applicable

		1	2	3	4	5	6	7	8	9	10	12	13	14	
$p_5 = x_3x_4$		-	-	1	-	1	-	-	-						
p^x_{5k}	K=1	-	-	0	-	1	-	-	-						
	2	-	-	1	-	0	-	-	-						
$p_6 = x_3x_6$		-	-	1	-	-	1	-	-						
p^x_{6k}	k=1	-	-	0	-	-	1	-	-						
	2	-	-	1	-	-	0	-	-						
$p_7 = x_4 \cdot x_5$		-	-	-	1	1	-	-	-						
p^x_{7k}	k=1	-	-	-	0	1	-	-	-						
	2	-	-	-	1	0	-	-	-						
$p_8 = x_4x_6$		-	-	-	1	-	1	-	-						
p^x_{8k}	k=1	-	-	-	0	-	1	-	-						
	2	-	-	-	1	-	0	-	-						
$p_9 = x_9x_{10}$										1	1	-	-	-	-
p^x_{9k}	k=1									0	1	-	-	-	-
	2									1	0	-	-	-	-
$p_{10} = x_9x_{11}$										0	-	1	-	-	-
p^x_{9k}	k=1									0	-	1	-	-	-
	2									1	-	0	-	-	-
$p_{11} = x_9x_{12}$										1	-	-	1	-	-
p^x_{11k}	k=1									0	-	-	1	-	-
	2									1	-	-	0	-	-

		1	2	3	4	5	6	7	8	9	10	11	12	13	14
$p_{12} = x_9 x_{13}$										1	-	-	-	1	-
p^x_{12k}	k=1									0	-	-	-	1	-
	2									1	-	-	-	0	-
$p_{13} = x_9 x_{14}$										1	-	-	-	-	1
p^x_{13k}	k=1									0	-	-	-	-	1
	2									1	-	-	-	-	0

<u>Efficiency</u>: For n = 22 inputs we have $3^n - 1$ multiple faults, i.e.

$$3^{22} - 1 = 3.138 \cdot 10^{10}.$$

All these faults are automatically covered by the lists for a-tests and b-tests.

6.3 Failure of a Residual Heat Removal System (RHR)

We have this System /21/, represented by a fault tree.
The undesired event is "RHR loss of isolation".

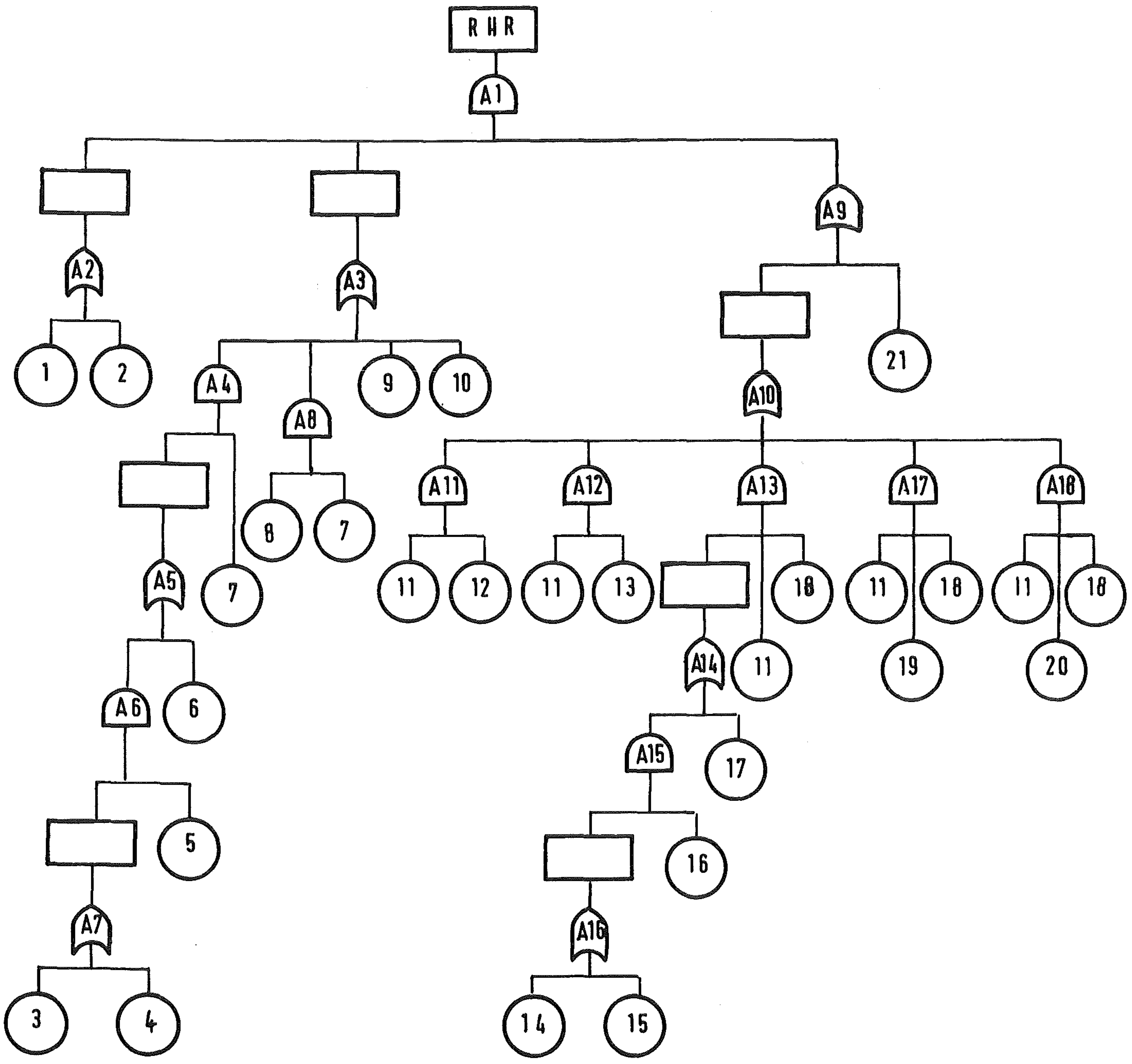


Fig. 29 RHR fault tree: restructured TOP.
(RHR, Residual Heat Removal)

The structure function is:

$\Phi = \quad A_2 \cdot A_4 \cdot A_{10}$	42 Min Cuts		
$+ A_2 \cdot A_8 \cdot A_{10}$	14 " "	= 84 Min Cuts	
$+ A_2 \cdot 9 \cdot A_{10}$	14 " "		
$+ A_2 \cdot 10 \cdot A_{10}$	14 " "		
$+ A_2 \cdot A_4 \cdot 21$	6 " "		
$+ A_2 \cdot A_8 \cdot 21$	2 " "	= 12 Min Cuts	
$+ A_2 \cdot 9 \cdot 21$	2 " "		
$+ A_2 \cdot 10 \cdot 21$	2 " "		
		96 Min Cuts	

where $A_2 = 1 + 2$

$A_4 = ((3 + 4)5 + 6) \cdot 7$

$A_8 = 7 \cdot 8$

$A_{10} = 11 \cdot (12 + 13 + ((14 + 15)16 + 17 + 19 + 20) \cdot 18)$.

For simplicity, we restrict the tests to A_{10}(F023 OPEN).

<u>Structure function for A_{10}</u>

A_{10} F023 OPEN

$= \quad A_{11} + A_{12} + A_{13} + A_{17} + A_{18}$

$A_{11} = 11 \cdot 12$

$A_{12} = 11 \cdot 13$

$A_{13} = A_{14} \cdot 11 \cdot 18$

$A_{14} = A_{15} + 17$ CONTROL SIGNAL TO F023

$A_{15} = A_{16} \cdot 16$

$A_{16} = 14 + 15$ INTERLOCK 2 PERMISSIVE

$A_{13} = 11((14 + 15)16 + 17)18$

$A_{17} = 11 \cdot 18 \cdot 19$

$A_{18} = 11 \cdot 18 \cdot 20$

$$\begin{aligned} A_{10} &= 11\cdot12+11\cdot13+11((14+15)16+17)18+11\cdot18\cdot19+11\cdot18\cdot20 \\ &= 11\cdot12+11\cdot13+11\cdot14\cdot16\cdot18+11\cdot15\cdot16\cdot18+11\cdot17\cdot18+11\cdot18\cdot19+11\cdot18\cdot20 \end{aligned}$$

a-Test

Min Cuts p_j	11	12	13	14	15	16	17	18	19	20
1	1	1	-	-	-	-	-	-	-	-
2	1	-	1	-	-	-	-	-	-	-
3	1	-	-	1	-	1	-	1	-	-
4	1	-	-	-	1	1	-	1	-	-
5	1	-	-	-	-	-	1	1	-	-
6	1	-	-	-	-	-	-	1	1	-
7	1	-	-	-	-	-	-	1	-	1

Minterms a_j	11	12	13	14	15	16	17	18	19	20	
1	1	1	0	0	0	0	0	0	0	0	A_{11}
2	1	0	1	0	0	0	0	0	0	0	A_{12}
3	1	0	0	1	0	1	0	1	0	0	
4	1	0	0	0	1	1	0	1	0	0	A_{13}
5	1	0	0	0	0	0	1	1	0	0	
6	1	0	0	0	0	0	0	1	1	0	A_{17}
7	1	0	0	0	0	0	0	1	0	1	A_{18}

$$\underbrace{\qquad\quad}_{A_{16}}$$

$$\underbrace{\qquad\qquad}_{A_{15}}$$

$$\underbrace{\qquad\qquad\quad}_{A_{14}}$$

Here is also information on subsystems (A_{16}, A_{15}, A_{14}) avaible.

We get more details than the mincuts alone.

b-Test

p_j and p^x_{jk}											Minterm b_{jk}									
	11	12	13	14	15	16	17	18	19	20	11	12	13	14	15	16	17	18	19	20
$p_1 = x_{11}x_{12}$	1	1	-	-	-	-	-	-	-	-										
p^x_{1k} k = 1	0	1	-	-	-	-	-	-	-	-	0	1	0	0	0	0	0	0	0	0
2	1	0	-	-	-	-	-	-	-	-	1	0	0	0	0	0	0	0	0	0
$p_2 = x_{11}x_{13}$	1	-	1	-	-	-	-	-	-	-										
p^x_{2k} k = 1	0	-	1	-	-	-	-	-	-	-	0	0	1	0	0	0	0	0	0	0
2	1	-	0	-	-	-	-	-	-	-	1	0	0	0	0	0	0	0	0	0
$p_3 = x_{11}x_{14}x_{16}x_{18}$	1	-	-	1	-	1	-	1	-	-										
p^x_{3k} k = 1	0	-	-	1	-	1	-	1	-	-	0	0	0	1	0	1	0	1	0	0
2	1	-	-	0	-	1	-	1	-	-	1	0	0	0	0	1	0	1	0	0
3	1	-	-	1	-	0	-	1	-	-	1	0	0	1	0	0	0	1	0	0
4	1	-	-	1	-	1	-	0	-	-	1	0	0	1	0	1	0	0	0	0
$p_4 = x_{11}x_{15}x_{16}x_{18}$	1	-	-	-	1	1	-	1	-	-										
p^x_{4k} k = 1	0	-	-	-	1	1	-	1	-	-	0	0	0	0	1	1	0	1	0	0
2	1	-	-	-	0	1	-	1	-	-	1	0	0	0	0	1	0	1	0	0
3	1	-	-	-	1	0	-	1	-	-	1	0	0	0	1	0	0	1	0	0
4	1	-	-	-	1	1	-	0	-	-	1	0	0	0	1	1	0	0	0	0
$p_5 = x_{11}x_{17}x_{18}$	1	-	-	-	-	-	1	1	-	-										
p^x_{5k}= k = 1	0	-	-	-	-	-	1	1	-	-	0	0	0	0	0	0	1	1	0	0
2	1	-	-	-	-	-	0	1	-	-	1	0	0	0	0	0	0	1	0	0
3	1	-	-	-	-	-	1	0	-	-	1	0	0	0	0	0	1	0	0	0
$p_6 = x_{11}x_{18}x_{19}$	1	-	-	-	-	-	-	1	1	-										
p^x_{6k} k = 1	0	-	-	-	-	-	-	1	1	-	0	0	0	0	0	0	0	1	1	0
2	1	-	-	-	-	-	-	0	1	-	1	0	0	0	0	0	0	0	1	0
3	1	-	-	-	-	-	-	1	0	-	1	0	0	0	0	0	0	1	0	0
$p_7 = x_{11}x_{18}x_{20}$	1	-	-	-	-	-	-	1	-	1										
p^x_{7k} k = 1	0	-	-	-	-	-	-	1	-	1	0	0	0	0	0	0	0	1	0	1
2	1	-	-	-	-	-	-	0	-	1	1	0	0	0	0	0	0	0	0	1
3	1	-	-	-	-	-	-	1	-	0	1	0	0	0	0	0	0	1	0	0

Efficiency: For $n = 21$ inputs we have 3^n-1 multiple faults, i.e.

$$3^{21} - 1 = 1.046 \cdot 10^{10} .$$

All these faults are automatically covered by the lists for a-tests and b-tests.

6.4 Nitric Acid Cooler

We consider a subsystem from chemical industry which cools in a process hot nitric acid (HNO_3) with a temperature feedback and a pump-shut-down feedforward. This has been analyzed by Lapp and Powers /22/.

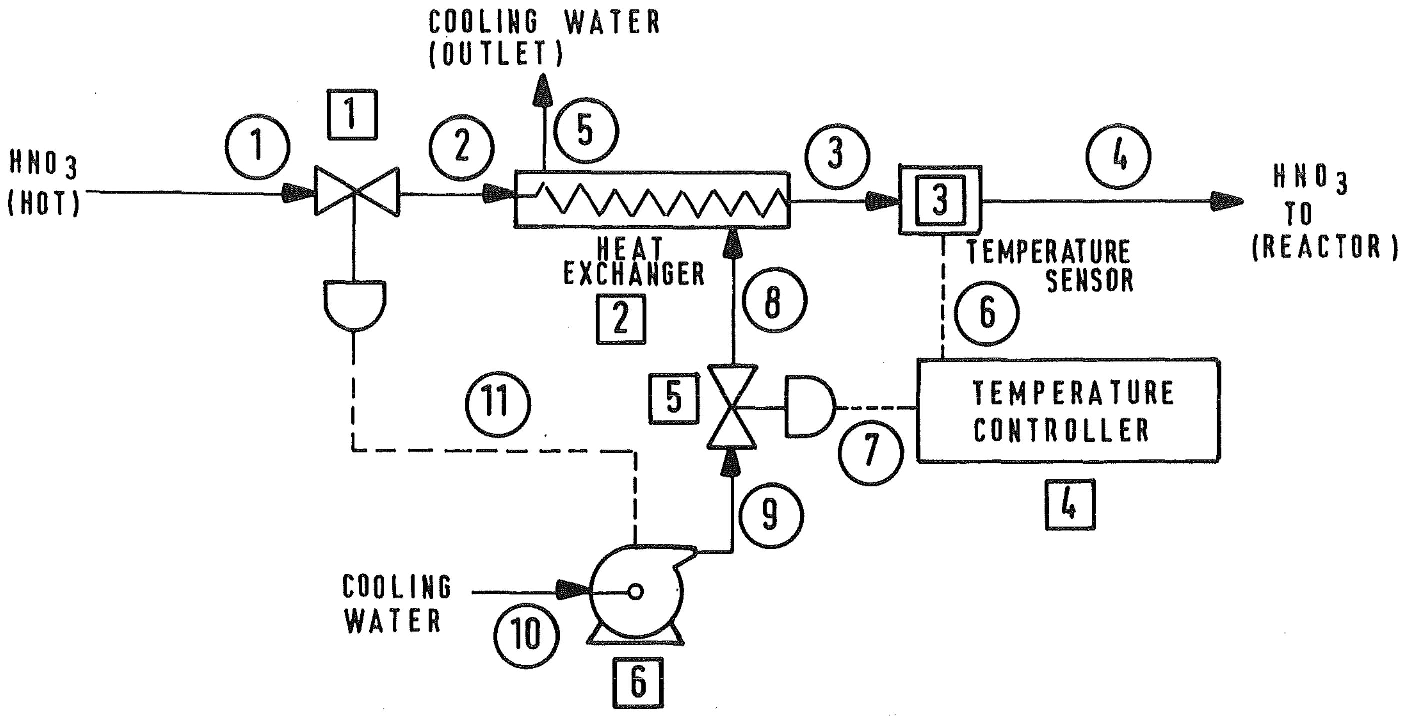


Fig. 30 Block diagram for nitric acid cooler

1. We list the components of this system giving:

   - possible inputs and outputs and
   - possible failures

   These may be translated into a fault table. But we will have a simpler way to deal with diagnosis by means of a-tests and b-tests.

2. Then we give a flow diagram for the possible processes including faults.

3. This leads to a non-coherent fault tree.

4. We get then the usual prime implicants and tests (again for a subtree).

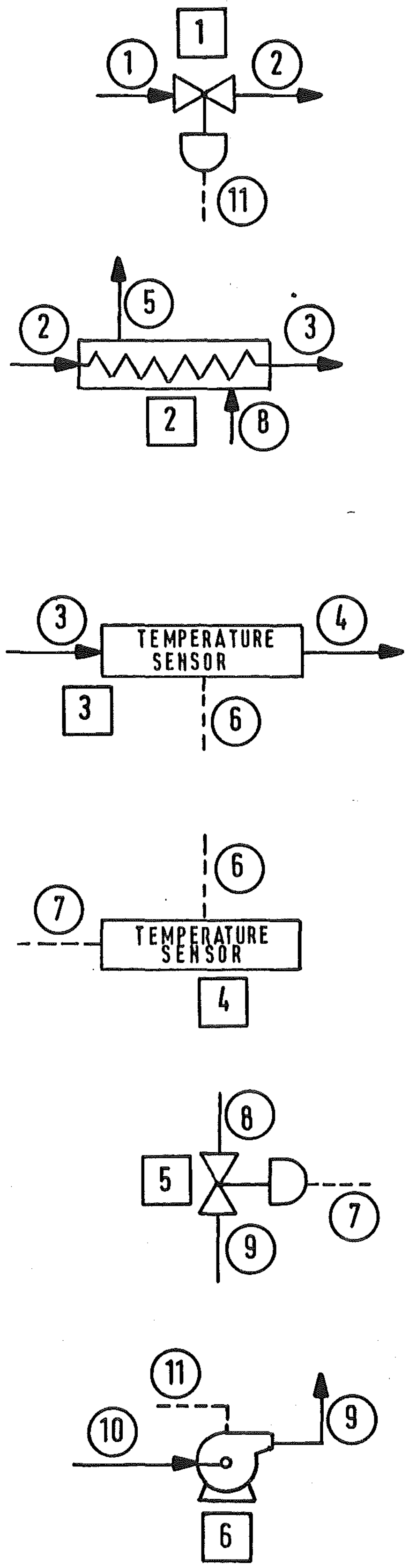


Fig. 31 Input-Output Models

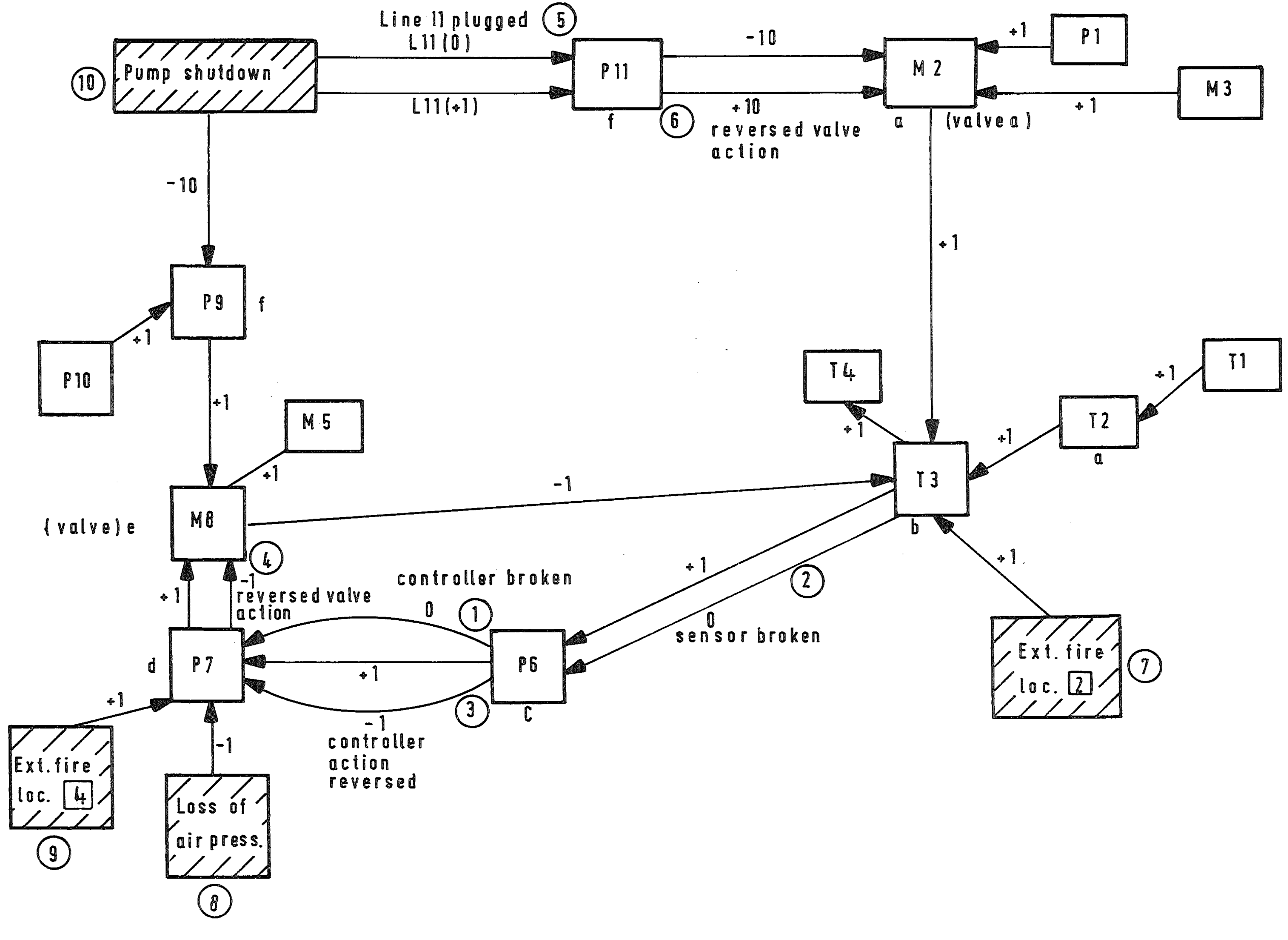


Fig. 32 Flow Diagram for Nitric Acid Cooler Process

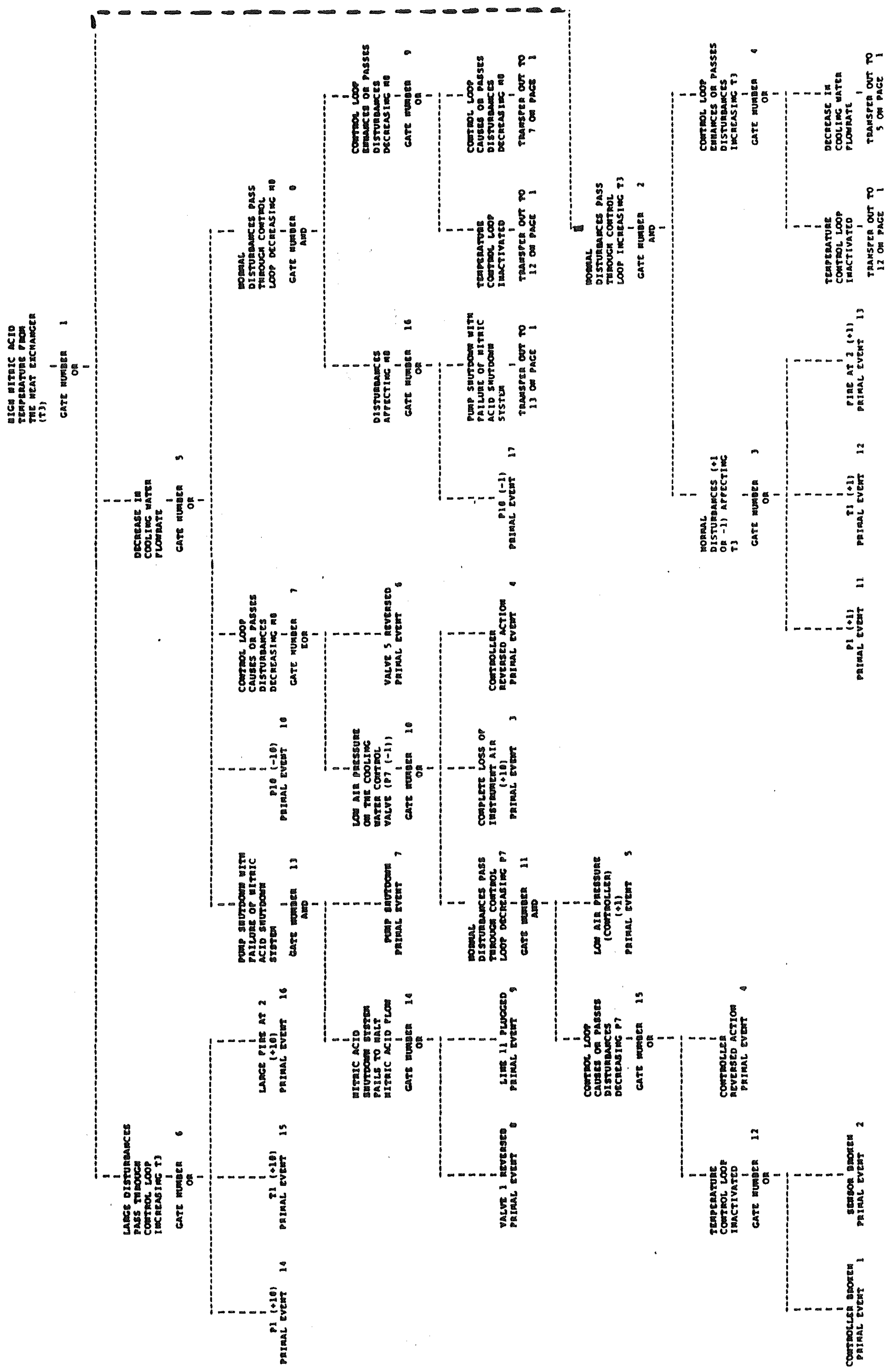

HIGH NITRIC ACID TEMPERATURE FROM THE HEAT EXCHANGER (T3) GATE NUMBER 1 OR
LARGE DISTURBANCES PASS THROUGH CONTROL LOOP INCREASING T3 GATE NUMBER 6 OR
P1 (+10) PRIMAL EVENT 14
T1 (+10) PRIMAL EVENT 15
LARGE FIRE AT 2 (+10) PRIMAL EVENT 16
DECREASE IN COOLING WATER FLOWRATE GATE NUMBER 5 OR
PUMP SHUTDOWN WITH FAILURE OF NITRIC ACID SHUTDOWN SYSTEM GATE NUMBER 13 AND
NITRIC ACID SHUTDOWN SYSTEM FAILS TO HALT NITRIC ACID FLOW GATE NUMBER 14 OR
VALVE 1 REVERSED PRIMAL EVENT 8
LINE 11 PLUGGED PRIMAL EVENT 9
PUMP SHUTDOWN PRIMAL EVENT 7
P10 (-10) PRIMAL EVENT 10
CONTROL LOOP CAUSES OR PASSES DISTURBANCES DECREASING M8 GATE NUMBER 7 EOR
LOW AIR PRESSURE ON THE COOLING WATER CONTROL VALVE (P7 (-1)) GATE NUMBER 10 OR
VALVE 5 REVERSED PRIMAL EVENT 6
NORMAL DISTURBANCES PASS THROUGH CONTROL LOOP DECREASING P7 GATE NUMBER 11 AND
COMPLETE LOSS OF INSTRUMENT AIR (+10) PRIMAL EVENT 3
CONTROLLER REVERSED ACTION PRIMAL EVENT 4
CONTROL LOOP CAUSES OR PASSES DISTURBANCES DECREASING P7 GATE NUMBER 15 OR
LOW AIR PRESSURE (CONTROLLER) (+1) PRIMAL EVENT 5
CONTROLLER REVERSED ACTION PRIMAL EVENT 4
TEMPERATURE CONTROL LOOP INACTIVATED GATE NUMBER 12 OR
CONTROLLER BROKEN PRIMAL EVENT 1
SENSOR BROKEN PRIMAL EVENT 2
NORMAL DISTURBANCES PASS THROUGH CONTROL LOOP DECREASING M8 GATE NUMBER 8 AND
DISTURBANCES AFFECTING M8 GATE NUMBER 16 OR
P10 (-1) PRIMAL EVENT 17
PUMP SHUTDOWN WITH FAILURE OF NITRIC ACID SHUTDOWN SYSTEM TRANSFER OUT TO 13 ON PAGE 1
CONTROL LOOP ENHANCES OR PASSES DISTURBANCES DECREASING M8 GATE NUMBER 9 OR
TEMPERATURE CONTROL LOOP INACTIVATED TRANSFER OUT TO 12 ON PAGE 1
CONTROL LOOP CAUSES OR PASSES DISTURBANCES DECREASING M8 TRANSFER OUT TO 7 ON PAGE 1
NORMAL DISTURBANCES PASS THROUGH CONTROL LOOP INCREASING T3 GATE NUMBER 2 AND
NORMAL DISTURBANCES (+1 OR -1) AFFECTING T3 GATE NUMBER 3 OR
P1 (+1) PRIMAL EVENT 11
T1 (+1) PRIMAL EVENT 12
FIRE AT 2 (+1) PRIMAL EVENT 13
CONTROL LOOP ENHANCES OR PASSES DISTURBANCES INCREASING T3 GATE NUMBER 4 OR
TEMPERATURE CONTROL LOOP INACTIVATED TRANSFER OUT TO 12 ON PAGE 1
DECREASE IN COOLING WATER FLOWRATE TRANSFER OUT TO 5 ON PAGE 1

Fig. 33 Fault Tree for Nitric Acid Cooler

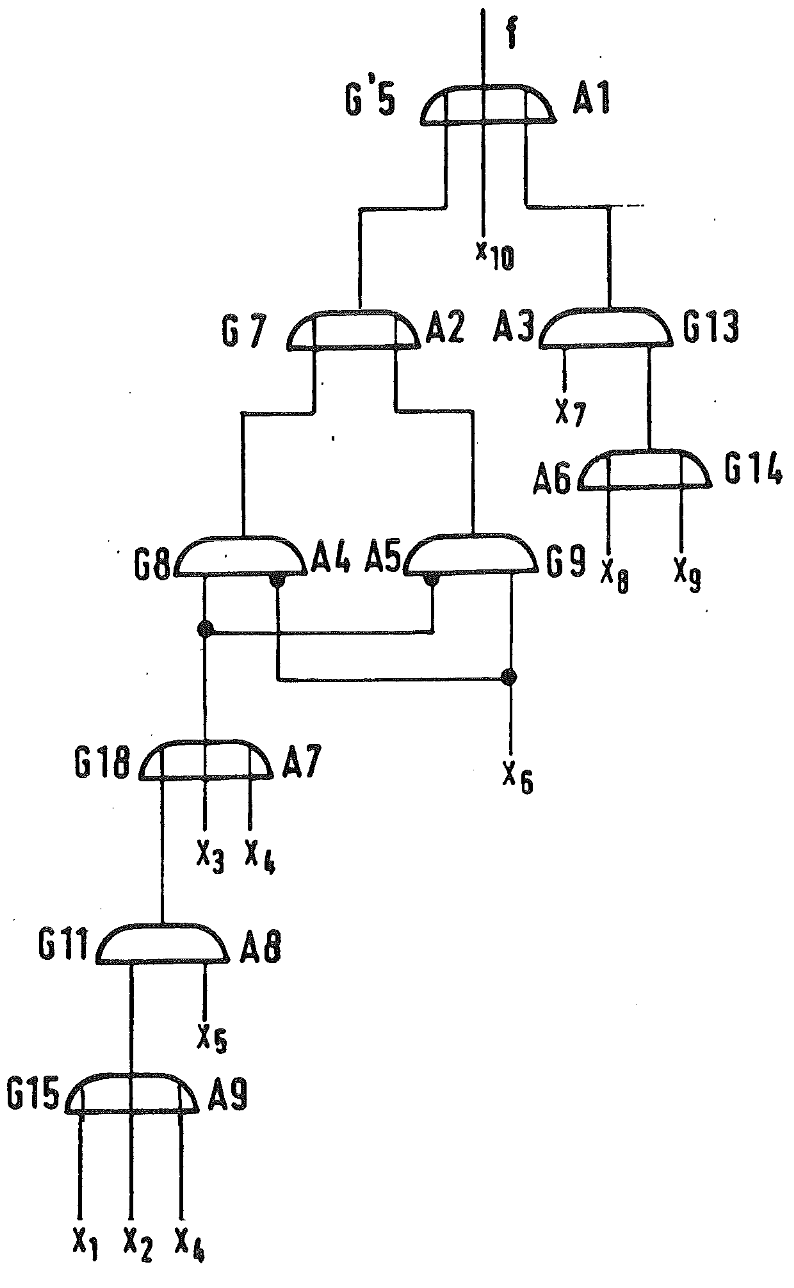


Fig.34 Subtree

Structure function of a non-coherent structure

(We use the top down algorithm, which here gives all prime implicants, but not for non-coherent structures in general.)

$$
\begin{aligned}
\Phi &= A_2 + x_{10} + A_3 \\
&= A_4 + A_5 + x_{10} + x_7 \cdot A_6 \\
&= A_7 \cdot \overline{x_6} + \overline{A_7} \cdot x_6 \qquad \text{(EXOR)} \\
&\quad + x_{10} + x_7 \cdot x_8 + x_7 \cdot x_9 \\
&= (A_8 + x_3 + x_4)\, \overline{x_6} + (\overline{A_3} \cdot \overline{x_3} \cdot \overline{x_4})\, x_6 \\
&\quad + x_{10} + x_7 \cdot x_8 + x_7 \cdot x_9 \\
&= ((x_1 + x_2 + x_4)\, x_5 + x_3 + x_4)\, \overline{x_6} \\
&\quad +((\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_4} + \overline{x_5})\, \overline{x_3} \cdot \overline{x_4})\, x_6 \\
&\quad + x_{10} + x_7 \cdot x_8 + x_7 \cdot x_9 \\
&= x_1 \cdot x_5 \cdot \overline{x_6} + x_2 \cdot x_5 \cdot \overline{x_6} + x_3 \cdot \overline{x_6} + x_4 \cdot \overline{x_6} \\
&\quad + \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} \cdot \overline{x_4} \cdot x_6 + \overline{x_3} \cdot \overline{x_4} \cdot \overline{x_5} \cdot x_6 \\
&\quad + x_{10} + x_7 \cdot x_8 + x_7 \cdot x_9
\end{aligned}
$$

a-Test

Prime Implicants

p_j	1	2	3	4	5	6	7	8	9	10
1	1	-	-	-	1	0	-	-	-	-
2	-	1	-	-	1	0	-	-	-	-
3	-	-	1	-	-	0	-	-	-	-
4	-	-	-	1	-	0	-	-	-	-
5	0	0	0	0	-	1	-	-	-	-
6	-	-	0	0	0	1	-	-	-	-
7	-	-	-	-	-	-	-	-	-	1
8	-	-	-	-	-	-	1	1	-	-
9	-	-	-	-	-	-	1	-	1	-

Minterms

a_j	1	2	3	4	5	6	7	8	9	10
1	1	0	0	0	1	0				
2	0	1	0	0	1	0				
3	0	0	1	0	0	0				
4	0	0	0	1	0	0				
5	0	0	0	0	1	1				
6	1	1	0	0	0	1				
7	0	0	0	0	0	0	0	0	0	1
8							1	1	0	0
9							1	0	1	0

b-Test for G5' of nitric acid cooler

		1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	
$p_1 = x_1 x_5 \bar{x}_6$		1 - - - 1 0 - - - -	32 16 8 4 2	*) Decimal
p^x_{1k}	k = 1	0 - - - 1 0 - - - -	0 1 0 0 1 0	18
	2	1 - - - 0 0 - - - -	1 0 0 0 0 0	32
	3	1 - - - 1 1 - - - -	1 0 0 0 1 1	35
$p_2 = x_2 x_5 \bar{x}_6$		- 1 - - 1 0 - - - -		
p^x_{2k}	k = 1	- 0 - - 1 0 - - - -	1 0 0 0 1 0	34
	2	- 1 - - 0 0 - - - -	0 1 0 0 0 0	16
	3	- 1 - - 1 1 - - - -	0 1 0 0 1 1	19
$p_3 = x_3 \bar{x}_6$		- - 1 - - 0 - - - -		
p^x_{3k}	k = 1	- - 0 - - 0 - - - -	0 0 0 1 0 0	4
	2	- - 1 - - 1 - - - -	0 0 1 0 0 1	9
$p_4 = x_4 \bar{x}_6$		- - - 1 - 0 - - - -		
p^x_{4k}	k = 1	- - - 0 - 0 - - - -	0 0 1 0 0 0	8
	2	- - - 1 - 1 - - - -	0 0 0 1 0 1	5
$p_5 = \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 x_6$		0 0 0 0 - 1 - - - -		
p^x_{5k}	k = 1	1 0 0 0 - 1 - - - -	1 0 0 0 0 1	33
	2	0 1 0 0 - 1 - - - -	0 1 0 0 0 1	17
	3	0 0 1 0 - 1 - - - -	0 0 1 0 1 1	11
	4	0 0 0 1 - 1 - - - -	0 0 0 1 1 1	7
	5	0 0 0 0 - 0 - - - -	0 0 0 0 1 0	2
$p_6 = \bar{x}_3 \bar{x}_4 \bar{x}_5 x_6$		- - 0 0 0 1 - - - -		
p^x_{6k}	k = 1	- - 1 0 0 1 - - - -	0 1 1 0 0 1	25
	2	- - 0 1 0 1 - - - -	1 0 0 1 0 1	37
	3	- - 0 0 1 1 - - - -	1 1 0 0 1 1	51
	4	- - 0 0 0 0 - - - -	1 1 0 0 0 0	48

*) We use a decimal numbering to check, if any of the b_{jk} is also included in more than one adjacent cubcubes. If this is not the case, all adjacent states can be identified.

		7	8	9	10	1	2	3	4	5	6	7	8	9	10
$p_7 = x_7x_8$		1	1	-	-										
p^x_{7k}	$k = 1$	0	1	-	-	0	0	0	0	0	0	0	1	0	0
	2	1	0	-	-	0	0	0	0	0	0	1	0	0	0
$p_8 = x_7x_9$		1	-	1	-										
p^x_{8k}	$k = 1$	0	-	1	-	0	0	0	0	0	0	0	0	1	0
	2	1	-	0	-	0	0	0	0	0	0	1	1	0	0

$p_9 = x_{10}$

Note: For p_9, which is a single failure, no adjacent subcubes exist.

Efficiency: For $n = 24$ inputs we have $3^n - 1$ multiple faults, i.e. $3^{24} - 1 = 2.824 \cdot 10^{11}$.
All these faults are automatically covered by the lists for a-tests and b-tests.

6.5 An illustrative Fault Tree

We are presenting a fault tree which has been already analyzed in sect. 1.7. (see /11/).
This fault tree is used for some research in simulation, where the system is not represented by software, but by hardware (e.g. with a s-o-p-representation, using diode logic /23/, /1/).
It is important to check this hardware in two respects:

- It is necessary to validate that the diode logic represents the original fault tree (This will not be discussed here).
- It is also necessary to test, whether there are any s-a-0 or s-a-1-faults in the diode logic. If there were any faults, this could seriously affect the simulation result.

Here is another, more direct application of the a-tests and b- tests.

The min-cuts for the following fault tree have been calculated by the bottom up algorithm (sect. 1.7).

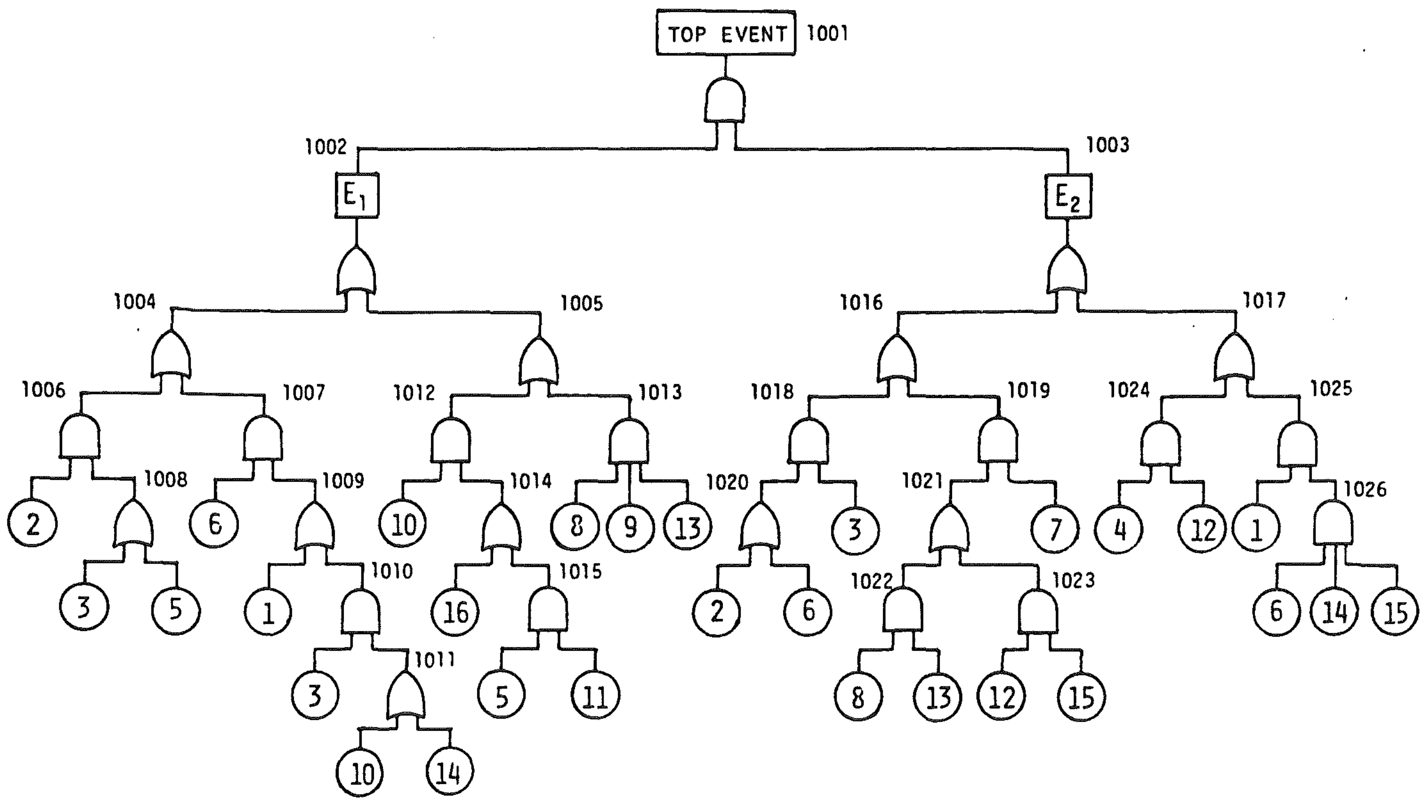


Fig. 35 Illustrative Example of Fault Tree.

Assume, we can get the outputs from E_1, E_2 separately. Then we get the following tests:

$\Phi_{E_1} = 2 \cdot 3 + 2 \cdot 5 + 1 \cdot 6 + 3 \cdot 6 \cdot 10 + 3 \cdot 6 \cdot 14 + 8 \cdot 9 \cdot 13 + 10 \cdot 16 + 5 \cdot 10 \cdot 11$

(similary we get Φ_{E_2}).

List

No.	min cut
1	2•3
2	2•5
3	1•6
4	3•6•10
5	3•6•14
6	8•9•13
7	10•16
8	5•10•11

<u>a-Tests</u> (subtree E_1)

Min Cuts P_j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	-	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-
2	-	1	-	-	1	-	-	-	-	-	-	-	-	-	-	-
3	1	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-
4	-	-	1	-	-	1	-	-	-	1	-	-	-	-	-	-
5	-	-	1	-	-	1	-	-	-	-	-	-	-	1	-	-
6	-	-	-	-	-	-	-	1	1	-	-	-	1	-	-	-
7	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	1
8	-	-	-	-	1	-	-	-	-	1	1	-	-	-	-	-

Minterms a_j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0
5	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
8	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0

Similary, we get a-tests for subtree E_2.

<u>b-Tests</u> (subtree E_1)

	p_j and p^x_{jk}																a_{jk}															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
p_1	-	1	1																													
	-	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-		0	1													
	-	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-		1	0													
p_2	-	1	-	-	1																											
	-	0	-	-	1												0	0			1											
	-	1	-	-	0												0	1			0											
p_3	1	-	-	-	-	1																										
	0	-	-	-	-	1											0					1										
	1	-	-	-	-	0											1					0										
p_4	-	-	1	-	-	1	-	-	-	1																						
			0	-	-	1	-	-	-	1									0			1				1						
			1	-	-	0	-	-	-	1									1			0				1						
			1	-	-	1	-	-	-	0									1			1				0						
p_5			1	-	-	1	-	-	-	-	-	-	-	1																		
			0	-	-	1	-	-	-	-	-	-	-	1					0			1								1		
			1	-	-	0	-	-	-	-	-	-	-	1					1			0								1		
			1	-	-	1	-	-	-	-	-	-	-	0					1			1								0		
p_6								1	1	-	-	-	1																			
								0	1	-	-	-	1											0	1				1			
								1	0	-	-	-	1											1	0				1			
								1	1	-	-	-	0											1	1				0			
p_7										1	-	-	-	-	-	1																
										0	-	-	-	-	-	1										0						1
										1	-	-	-	-	-	0										1						0
p_8					1	-	-	-	-	1	1																					
					0	-	-	-	-	1	1										0					1	1					
					1	-	-	-	-	0	1										1					0	1					
					1	-	-	-	-	1	0										1					1	0					

Similarly, we get b-tests for subtree E_2.

Efficiency: For $n = 20$ inputs we get $3^n - 1$ multiple faults, i.e.

$$3^{20} - 1 = 3.487 \cdot 10^9.$$

All these faults are automatically covered by the lists for a-tests and b-tests.

References

/1/ Z. Kohavi, Switching and Finite Automata Theory
Mc Graw-Hill Book Company, New York 1978

/2/ J. P. Hayes, Computer Architecture and Organization
Mc Graw-Hill Book Company, New York 1978

/3/ K. E. Iverson, A Programming Language
John Wiley and Sons Inc., New York 1962

/4/ B. Girling, H.G. Moring
Logic and Logic Design
Intertext Books, International Texbook
Company Limited, 1973

/5/ V. T. Rhyne, et al.
A new Technique for the Minmization of Switching Functions
IEEE-Trans. on Computers
Vol. C-26, pp. 757 -763 (1977)

/6/ M. Davio, J.-P. Deschamps, A. Thayse
Discrete and Switching Functions
Mc Graw-Hill Book Company,
New York 1978

/7/ R. J. Nelson, Simplest Normal Truth Functions
J. Symbolic Logic, Vol. 20 pp. 105-108, (1954)

/8/ B.L. Hulme, R. B. Worrell
A Prime Implicant Algorithm with Factoring
IEEE-Trans. on Computers
Vol. C-24, pp. 1129-1131 (1975)

/9/ J. B. Fussell, W. E. Vesely
A new Methodology for obtaining
Cut Sets for Fault Trees
Trans. Amer. Nucl. Soc., Vol. 15, pp. 262-263,
June 1972

/10/ R. G. Bennetts
On the Analysis of Fault Trees
IEEE Trans. on Reliability
Vol. R-24, pp. 175-185 (1975)

/11/ K. Nakashima, Y. Hattori
An Efficient Bottom-up Algorithm for Enumerating
Minimal Cut Sets of Fault Trees
IEEE-Trans. on Reliability, Vol. R-28 pp. 353-357
(1979)

/12/ M. A. Breuer, A. D. Friedman,
Diagnosis & Reliable Design of Digital Systems
Pitman Publ. Ltd., London, 1977

/13/ J. D. Murchland, G. G. Weber
A Moment Method for the Calculation of a Confidence
Interval for the Failure Probability of a System
Proceedings of 1972 Annual
Reliability and Maintainability Symposium, San Francisco,
pp. 565-577

/14/ R. E. Barlow, F. Proschan
Statistical Theory of Reliability and Life Testing
(Probability Models)
Holt, Rinehart and Winston Inc., New York, 1975

/15/ U. Höfle-Isphording
Zuverlässigkeitsrechnung
Springer Verlag, Berlin, 1978

/16/ VDI Richtlinie 4008/Blatt 7
Strukturfunktion und ihre Anwendung
(Entwurf), Verein Deutscher Ingenieure, Düsseldorf 1979

/17/ S. C. Lee, Modern Switching Theory and Digital Design
Prentice-Hall Inc., Englewood Cliffs,
New Jersey, 1978

/18/ I. Kohavi, Z. Kohavi,
Detection of Multiple Faults on Combinational
Logic Networks
IEEE-Trans. on Computers, Vol. C-21, pp. 556-568
(1972)

/19/ G. G. Weber
Untersuchung des Zusammenhangs zwischen Fehlerbaumanalyse
und Störfallanalyse am Beispiel des Photometer-Leitfähig-
keitsmeßstandes, KfK 2909, Februar 1980,
Kernforschungszentrum Karlsruhe

/20/ D. Stöckle,
Unpublished Results

/21/ H. E. Lambert
Fault Trees for Decision Making in Systems Analysis
(Ph. D. - Thesis), UCRL-51829, University of California,
Livermore, 1975

/22/ S. L. Salem, G. E. Apostolakis, D. Okrent
A new Methodology for the Computer-Aided
Construction of Fault Trees
Ann. of Nucl. Energy, Vol. 4, pp. 417-433,
Pergamon Press 1977

/23/ S. A. Lapp, G. J. Powers
Computer Aided Synthesis of Fault Trees
IEEE-Trans. on Reliability, Vol. R-26, pp. 2-13, 1977
and
S. A. Lapp, G. J. Powers
Update of Lapp-Powers Fault-Tree Synthesis Algorithm
IEEE-Trans. on Reliability, Vol. R-28, pp. 12-15, 1979

/24/ S. Fenyi
Unpublished Results

/25/ K. Nakashima
Studies on Reliability Analysis and Design of
Complex Systems,
PhD- Thesis, KYOTO UNIVERSITY, Kyoto Japan, March 1980

/26/ W. Görke,
Generating Tests for Functional Expressions in
Self-Diagnoses and Fault-Tolerance, Proceedings,
Mario Dal Cin, Elmar Dilger (Eds.)
Attempo Verlag, Tübingen 1981